National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

NL-339 (r. 88/04) c

Canada

University of Alberta

A Partial-Destination-Release Strategy for the Multi-Token Ring Network

by

May-Yew Wee

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Master of Science

Department of Computing Science

Edmonton, Alberta
Spring 1992

National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-73139-7

Canada

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: May-Yew Wee
TITLE OF THESIS: A Partial-Destination-Release Strategy
for the Multi-Token Ring Network

DEGREE: Master of Science
YEAR THIS DEGREE GRANTED: 1992

Permission is hereby granted to the UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

Permanent Address:
82A Lorong J
Telok Kurau
Singapore 1542
Republic of Singapore

Date: April 13th, 1992

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled A Partial-Destination-Release Strategy for the Multi-Token Ring Network submitted by May-Yew Wee in partial fulfillment of the requirements for the degree of Master of Science.

. . . . . .

A. Kamal (Supervisor)

. . . . . .

P. Gburzynski

. . . . . .

W. Krzymien

Date: April 13th, 1992

To my family, the Durands
and Bailey

Abstract

This thesis defines a new protocol using multiple tokens in a token ring network where tokens are released either at their sources or at their destination nodes, depending on bandwidth availability. The new protocol is able to handle multiple priority levels in the transmission of packets with minimal added computation at the nodes. The problem of fairness in the distribution of bandwidth is also addressed and a fairness mechanism that can be implemented at the expense of some degradation in performance is introduced. A new feature is also proposed where the number of tokens can be changed dynamically. This feature can be implemented on top of the priority handling procedures with relative ease and with proper definition of the priority field semantics.

In token ring networks, the reliability issue is an important one, simply because the topology of the network is vulnerable to single node failures. The main reliability issue discussed in this paper is error detection and how modifications to this protocol will improve the reliability of the network.

Then, a simulated model of the protocol is used to study the performance of the protocol. The protocol is shown to outperform several other popular protocols.

Acknowledgements

I would like to express my deepest thanks to my supervisor, Dr. Ahmed Kamal for his continued guidance and encouragement in the writing of this thesis. His knowledge and insight has continually challenged me to achieve even better results and I am extremely grateful for that. I would also like to thank the committee members, Dr. P. Gburzynski and Dr. W. Krzymien for their comments and suggestions. Although my family is not here with me, their concern and care has provided me with the motivation to work hard and give my best to this thesis. A group of people whom I owe my deepest gratitude is the Alberta Shitoryu Karate Club of Canada. Their support, friendship and concern has made my stay here in Edmonton most memorable and provided me with the drive to do well in my studies. There is also my roommate, Lorraine Durand whom I am grateful for putting up with me for the past few months, and vice versa. Last of all, I would like to thank Bailey for his companionship and devotion, through the good and the bad times.

Contents

1 Introduction **1**

1.1 Background 1

1.2 Motivation and Contribution 11

1.3 Thesis Outline 14

2 Description of Partial-Destination-Release Multi-token Protocol **16**

2.1 Introduction 16

2.2 The PDR Protocol 17

2.2.1 Node Table 20

2.2.2 Token Table 21

2.2.3 Transmitter Process 23

2.2.4 Receiver Process 30

2.2.5 Table Management Process 32

2.2.6 Example 34

2.3 Overhead 42
2.4 Fairness 43
2.4.1 DSDR protocol 44
2.4.2 Fairness mechanism in Counter Rotating Rings 46
2.4.3 Orwell protocol 48
2.4.4 PDR protocol 49
2.5 Reliability 51

3 Study of New Protocol through Simulation Model 54

3.1 Introduction 54
3.2 Simulation Model 55
3.3 Performance Measure 57
3.4 Comparison to Other Protocols 60
3.5 Implementing fairness 69
3.6 Upper bounds 73
3.7 Summary 74

4 Priority Handling 75

4.1 Introduction 75
4.2 IEEE 802.5 protocol 76
4.3 Timer-based Priority protocols 76
4.4 A Modified Reservation-based Priority protocol 78
4.5 A Packet Preemption protocol 80

4.6 Priority handling in PDR protocol 80
4.7 Performance Measure 84
4.8 Summary 86

5 Dynamic Changing of the Number of Tokens **88**
5.1 Introduction 88
5.2 Protocol Description 96
5.3 Simulation Results 101
5.4 Summary 105

6 Conclusions **107**

References **112**

List of Tables

2.1 The 8 different states of a token indicated by the status field . 20

3.1 Bandwidth Utilization and Maximum Transfer Delay 71

List of Figures

1.1 The four basic LAN topologies 3

1.2 High speed ring network with $a > 1$ 6

1.3 High speed ring network with $\frac{a}{2} \gg 1$ 13

2.1 Token format for Start transmission 18

2.2 Token format for all other states 18

2.3 Node Table 21

2.4 Token Table 22

2.5 Algorithm for Transmitter Process 26

2.6 Algorithm for Receiver Process 31

2.7 Algorithm for Table Management Process 33

2.8 Token Format with FCS 52

3.1 PDR protocol with 1, 2, 4, 8, 12 and 16 tokens 58

3.2 Source and Destination release protocols 63

3.3 Comparison with Slotted ring and DQDB: R=1000, m=500 (exp) 65
3.4 Comparison with Slotted ring and DQDB: R=5000, m=1000 (exp) 65
3.5 Comparison with Slotted ring and DQDB: R=2000, m=500 (exp) 66
3.6 Comparison with Slotted ring: R=2000, m=160 (constant) . . 68
3.7 Comparison with DQDB: R=5000, m=1184 (constant) 69
3.8 PDR protocol with and without fairness 70
3.9 Distribution of bandwidth among the stations 72

4.1 Algorithm for Priority Handling 77
4.2 Priority scheme with 2 priority levels using 2 and 8 tokens . . 85
4.3 Preemtable and non-preemtable priority schemes with 2 and 8 tokens 85

5.1 PDR protocol with 4, 8, 13 and 16 tokens: R=1000, m=200 (exp) 92
5.2 PDR protocol with 2, 6, 10 and 14 tokens: R=1000, m=400 (exp) 93
5.3 PDR protocol with 5, 10, 17 and 20 tokens: R=1000, m=100 (exp) 94

5.4 PDR protocol with 10, 15, 26 and 30 tokens: R=2000, m=200 (exp) 95
5.5 Algorithm for addition/removal of tokens in Receiver process . 99
5.6 Algorithm for addition/removal of tokens in Table Management process 100
5.7 PDR-DC protocol with window size = $50R$ 104
5.8 PDR-DC protocol with window size = $200R$ 105

Chapter 1

Introduction

1.1 Background

Communication among devices such as computers, terminals, peripheral devices etc., has become a necessity that follows from a desire to share information and resources. Such devices are often referred to as nodes, stations, sources, etc. A communication network provides the facility to do so by providing an interconnection mechanism and a set of rules to be used for the exchange of information over this mechanism. A Local Area Network (LAN) is a special case of communication networks in which the network covers a limited geographical area (a span of only several kilometres). A Wide Area Network (WAN) is the other extreme network that covers a much wider domain of several thousand kilometres. In between these two extremes, there

is the Metropolitan Area Network (MAN), which is basically an interconnection of several LANs. For a further description of network classifications and their characteristics, the reader is referred to [25].

LANs can be classified according to their topology and their medium access control technique. The four basic types of LAN topologies are: ring, bus, star and tree (Figure 1.1). Communication in the ring topology is unidirectional and for the other three topologies it is usually bidirectional. The ring topology is one of the most popular topologies. The three most common medium access techniques for ring networks are: token ring, slotted ring and register-insertion ring network protocols.

In the Token Ring protocol [10], access to the communication medium is controlled by a token. The token circulates in the ring and has a status bit to indicate if it is empty or full. Permission to transmit a packet is granted to the station that sets the token status from empty to full. At the end of its packet transmission or when the token returns after propagating one round trip, whichever comes later, the station resets the token status back to empty and passes the token on to the next station. The Token Ring is one of the simplest access protocols to implement. It has also been chosen as the IEEE 802.5 standard for LANs.

In the slotted ring protocol [9], the communication medium is divided into multiple slots of fixed and equal lengths. Each slot has a status bit to indicate if the slot is empty or full. To claim an empty slot for packet

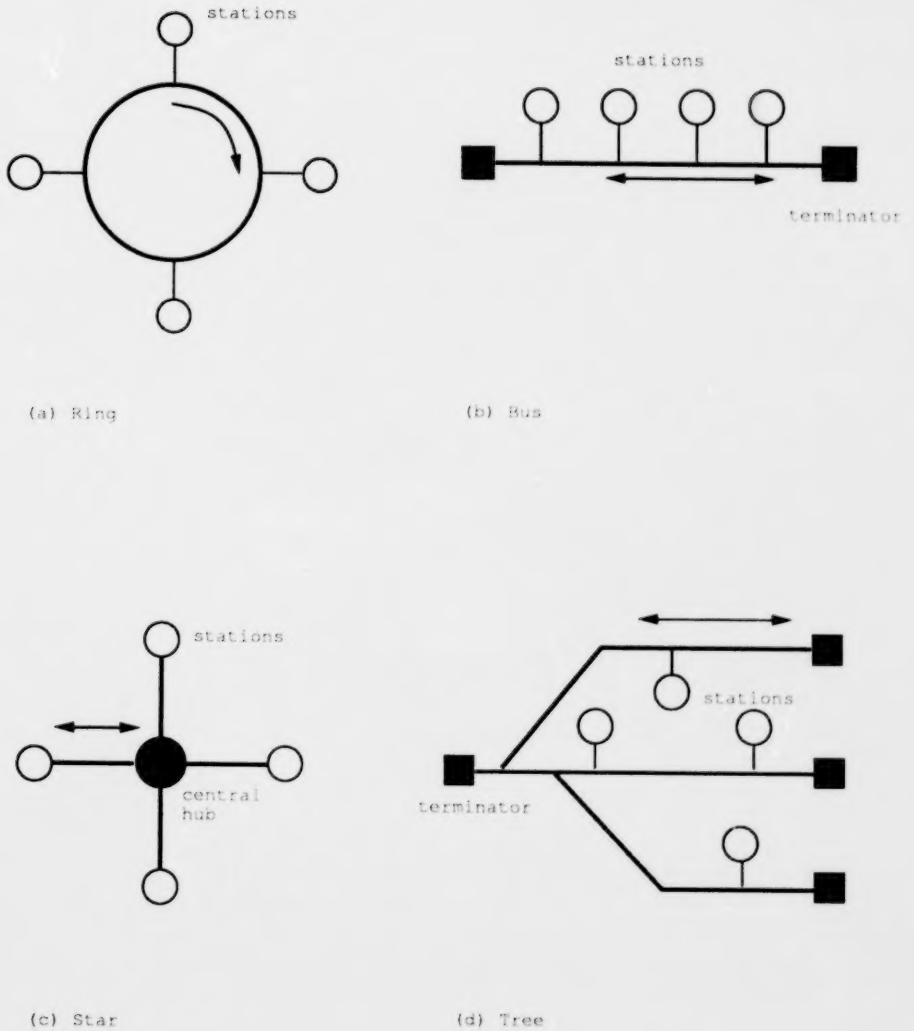


Figure 1.1: The four basic LAN topologies

transmission, a station has to mark the slot status as full and insert its data within the slot length. When the same slot returns after one round trip, the slot status is marked empty again. Since the slots are of fixed lengths, packets longer than the slot will have to be divided into smaller subpackets to fit into the slots.

In the register-insertion ring [7], a station can transmit a packet as long as the communication medium is free and the amount of free buffer space at the station is greater than or equal to the size of the packet to be transmitted. While a station is transmitting its packet, any upstream traffic detected on the medium is stored in the station's buffer. At the end of its own packet transmission, the station will proceed to transmit the data stored in its buffer and will empty the buffer upon its packet's return. Access control is therefore asynchronous and distributed. An extension of this protocol that allows the buffer to be emptied when idle bits are encountered has been introduced in [22].

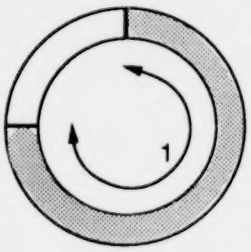
Recent developments in computer technology have resulted in very powerful microprocessors that can handle greater amounts of storage and data processing. The exchange of such large amounts of data at such high processing rates has motivated the use of fiber-optic technology in LANs, in place of the slower media such as coaxial cables and twisted pairs. With optical fibers, data transmission rates of several hundreds of millions of bits per second can be achieved compared to only tens of megabits per second for

coaxial cables and the few megabits per second for twisted pairs.

This has led to the emergence of High Speed LANs (HSLAN). Such LANs cannot operate satisfactory using network protocols that were developed for slower LANs. Due to the high data transmission rates, packet transmission delays are comparatively much smaller than the ring propagation delays. Under such circumstances, the performance of the traditional 802.5 Token Ring protocol would be adversely affected by the ring propagation delay which results in rapid performance degradation as either the span of the network or the transmission rate increases.

To benefit instead of being hindered by this tremendous increase in data rates, the Early Token Release (ETR) medium access control technique was proposed. The ETR protocol is based on a token-passing control technique similar to the Token Ring, but instead of waiting for the header to return before releasing the token (as in the Token Ring), ETR releases the token immediately after the end of packet transmission. Accordingly, although only one station is actively transmitting a packet at any one time, there may be multiple packets propagating on the ring on their way to their destinations at the same time. Thus channel utilization is not limited by the network size and a higher value can be achieved. The 802.5 Token Ring protocol on the other hand, has only one "*active*" packet[1] utilizing the channel at any one

[1]The packet is propagating on the ring on its way to its destination.



$a > 1$ where
a = total ring delay
1 = packet length

Figure 1.2: High speed ring network with $a > 1$

time.

Consider a high speed ring network with N stations. To give an easy representation of the total ring delay in relation to the packet length, we define the normalized total ring delay, a, as,

$$a = \frac{\text{total ring delay}}{\text{packet length}}$$

the normalized packet length is therefore equal to 1 (Figure 1.2). The maximum achievable throughput (T) of the IEEE 802.5 Token Ring is given by:

$$T = \frac{1}{a + \frac{a}{N}} = \frac{1}{a(1 + \frac{1}{N})} \tag{1.1}$$

where $a > 1$. Note from the equation that a varies inversely proportional to T. The value of a increases as the network size or the data transmission rate increases. For the ETR protocol, the maximum achievable throughput is given by:

$$T = \frac{N}{N + a} = \frac{1}{1 + \frac{a}{N}} \tag{1.2}$$

for any value of a. Note that in this case the value of a has a much lesser impact on the maximum achievable throughput.

However, in single token protocols, the network size still continues to have an important influence on the access delay. With only one token operating in the ring, the average time that a station has to wait for the arrival of the token under very light load is approximately equal to half the total ring delay. When the size of the ring is large, this waiting time also becomes significantly large. To reduce the influence of the ring size on performance, a multiple-token ring protocol was proposed in [11]. The access control technique is similar to that of the ETR except that now there are multiple tokens physically present in the ring. The waiting time for a token to visit a station is obviously reduced, especially at light loads. However, due to the increased overhead of having more tokens, the maximum achievable throughput can be slightly less than that of ETR, especially if a large number of tokens is used.

The slotted and register-insertion ring protocols can be seen as variations to the multiple token ring protocol, but with differing access strategies. In slotted rings [9], the multiple access points are found at fixed locations, hence timing at different stations have to be synchronized. Fragmentation in slotted rings may occur since packets of various lengths may have to be transmitted using the fixed length slots, thus resulting in partially filled slots. In the register-insertion ring [7], a buffer is required at each station to hold incoming traffic. A station can transmit as long as the buffer space available is large enough to hold any incoming traffic while its own packet is being transmitted. Hence the buffer has to be large enough to hold the largest packet size allowed.

The token-passing technique has been preferred over these two ring protocols because the medium access control can ensure a bounded access delay, a fair distribution of bandwidth, a simple handling of priorities, the ability to recover from single node failures rather easily, and above all a reasonably good performance.

The protocols mentioned so far are all source release protocols. That is, the transmitted data propagates once around the ring before it is removed by the source node. There are several advantages associated with source release token ring protocols:

- The destination node is able to piggyback acknowledgements on the received packet since the packet has to propagate back to the source node.

- Since a station has to release all used tokens before transmitting the next packet, a fair distribution of bandwidth among the stations is ensured. This also sets an upper bound on the access time.

- No delay is required at the nodes to decide if it is the intended recipient of a packet because the information can be read from the token header on the fly.

The only disadvantage of source release protocols is that even after a packet has been received by the destination node, the packet continues to occupy

bandwidth as it propagates back to the source. If the destination of a packet is uniformly distributed, then on the average, half of the total ring bandwidth is being unneccessarily occupied by *"old and useless"* information. Hence, a common method used to improve channel utilization is to have packets removed by their destinations. With destination release, the advantages associated with source release protocols no longer hold.

Both the Pierce [20] and the Orwell [1] protocols employ the destination release technique in a slotted ring network. The amount of in-line buffering delay required at each node is equivalent to the length of the destination address field. To solve the problem of fairness in bandwidth distribution, a fairness mechanism has to be implemented into the protocol. This will be explained in detail in Chapter 2.4.

A protocol proposed by [12] also employs destination release in a slotted ring but only in a partial manner. The first and the last slots occupied by a packet must still be released by the source node. In this protocol, only the first slot used to transmit the packet contains the address fields. By allowing the first slot to be released by the source, the delay at each node for reading the destination address is therefore dispensable. Instead, it is replaced by a 2-bit delay to read the slot status. The reason for having the last slot released by the source is twofold: first to enable the destination node to piggyback its acknowledgement back to the source, and second to inform the rest of the nodes of the termination of that communication session. The reader is

refered to [12] for a detailed description of the protocol.

One protocol that employs destination release in a Token Ring network is the Concurrent Token Ring protocol proposed by [28]. It is based on partial destination release in the sense that only the data field of the frame (which includes the trailer that marks the end of the data) is removed at the destination. At the destination, the node changes the token to a *Cut* frame by setting a status bit, removes the data field and appends an empty token right after the *Cut* frame. The token headers continue to propagate back to their sources in the form of *Cut* frames to be removed. It is thus possible to find multiple *Cut* frames in front of a token if the token has been used and then released more than once within one round trip. A node wishing to transmit can do so if the destination of its packet is before the source of the first *Cut* frame. This is because the source of the first *Cut* frame might still be in the process of a packet transmission. In other words, a station can only transmit its packet within the unused portion of the ring. Additional buffer space is required at each station to store *Cut* frames that may arrive while the station is still transmitting. These *Cut* frames will be stored in the station's buffer until the station's own packet transmission has been completed, then the stored *Cut* frames will be passed on to the next station. This method does not require any added delay at all to read the address fields but the problems of acknowledgment and fairness still remain.

1.2 Motivation and Contribution

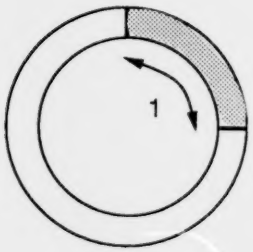
With the issue of designing efficient HSLANs becoming more important, the need for protocols that can better utililize and benefit from the high data rates has become more evident. Various protocols with multiple access points have been proposed for high speed ring networks. Protocols like the Cambridge Fast Ring [27], [8], the Orwell Ring [1], slotted ring protocols by [17], [12] and [14] all employ the slotted ring medium access technique. However there are certain problems associated with the slotted ring access technique that limit its performance and capabilities:

- The use of variable length packets may result in fragmentation which directly affects the performance of the protocol. The severity of fragmentation is dependent on the relation between the slot length and the packet length.
- The number of slots in the ring is dependent on the total ring delay which makes it more difficult to add/remove nodes or dynamically add/remove slots, as compared to the token-passing technique.
- The access delay is unbounded.
- There is no equitable distribution of bandwidth among the stations.
- Synchronization is required at all the nodes.

The only multiple access point protocol that employs the token-passing technique is the multiple token ring protocol proposed by [11]. It has been shown to outperform the ETR at light and medium loads, but is slightly worse at heavy loads. The goal of the new protocol proposed in this thesis is to extend and enhance the performance of the multiple token ring protocol at all load levels. A better utilization of ring bandwidth is achieved by adopting the partial destination release technique proposed in [28]. The new protocol uses the same concept of partial destination release, but with a different implementation. This change in implementation is neccessary because the new protocol is defined for high speed networks while the Concurrent Token Ring (CTR) protocol by [28] is designed for the slower LANs, viz., the IEEE 802.5 Token Ring.

Consider a ring network with the normalized total ring delay equal to a and the normalized packet length equal to 1. Assuming that the destination of a packet is uniformly distributed, the average time for a packet to reach its destination is equal to $\frac{a}{2}$. Therefore in the CTR protocol, the average service time [2] is equal to $\frac{a}{2}$, regardless of the packet length. In high speed networks, it is possible to have a large value of a such that $\frac{a}{2} \gg 1$ (Figure 1.3). In such a situation, the service time is reduced if the token can be released immediately after the end of packet transmission. Hence in the new

[2]The time between the capture and the release of the token.



$\frac{a}{2} \gg 1$ where
$\frac{a}{2}$ = distance between source and destination
1 = packet length

Figure 1.3: High speed ring network with $\frac{a}{2} \gg 1$

protocol, a token can be released by either the source or the destination node depending on the available bandwidth, as will be explained later.

Other than solely striving for better performance, problems such as fairness, priority handling and reliability are also addressed. The contribution of this thesis is as follows:

1. A new protocol is proposed for high speed token ring networks that can outperform the multiple token ring protocol by [11] and also some of the popular network access protocols. The performance superiority of the new protocol is achieved over almost the entire range of traffic loads, with varying packet lengths and different network spans. The new protocol is also able to handle priority transmissions of packets based on the same, simple procedure employed in the IEEE 802.5 Token Ring.

2. As in all destination release protocols, there is unfair distribution of bandwidth among the stations in the new protocol. To overcome this problem, a new and efficient fairness mechanism that requires minimal added overhead and computation is proposed.

3. Performance of the new protocol will improve as the number of tokens in the network is increased, but only up to a certain threshold. Beyond this threshold, further increases in the number of tokens will result in rapid degradation of the network performance. A procedure is defined to estimate the value of this threshold.

4. A new feature is proposed that allows the network to dynamically change the number of tokens in the ring based on the current network parameters. This feature can be implemented in the new protocol by incorporating it into the existing priority handling procedures with only a small amount of added computation and overhead.

1.3 Thesis Outline

In the next chapter, a detailed description of the new protocol is given together with a discussion and solution for fairness in bandwidth distribution. Chapter 3 shows the simulation results of our protocol compared to other protocols. These include figures for the bandwidth distribution when the fairness mechanism is implemented. The fourth chapter concentrates solely on the handling of priority transmissions. A description of well known priority protocols is also provided. Again, using our simulation model, we study the amount of service provided to the different levels of priority. Chapter 5 is a proposal for an interesting feature to be added to the new protocol: the

dynamic changing of the number of tokens in the ring. The motivation for this feature is based on simulation results that show that the best performance of the protocol can be achieved by a number of tokens that changes with changing network parameters and workload conditions. The thesis is then rounded up with a brief overview of the results and conclusions that we have obtained, together with proposals for further investigations in areas which can be improved or extended.

Chapter 2

Description of Partial-Destination-Release Multi-token Protocol

2.1 Introduction

The objective of having multiple tokens circulating in a ring is to improve bandwidth utilization by providing multiple access points to the ring medium. However there are two main problems associated with multiple tokens circulating in a ring.

The first problem is the prevention of collisions as more than one node may be transmitting at the same time. Hence a transmitting node must

know when the next token arrives so it can put its own data transmission on hold to prevent a collision with the arriving upstream traffic.

The second problem is the overhead involved when a packet needs to be broken up into smaller sub-packets to fit into the available bandwidth. For the destination node to receive and reassemble the packets, it needs to know the source and destination addresses of every sub-packet transmitted.

To solve these two problems, an approach similar to the multiple token protocol described in [11] is used. The motivation for the Partial-Destination-Release Multi-token (PDR) protocol is to improve on the performance of the multiple token algorithm proposed in [11] by having tokens released at their destinations instead of at their sources. The intent is also to extend the protocol to handle priority transmissions and dynamic changing of the number of tokens in the ring.

In this section, a detailed description of the access protocol will be given which will include a discussion of overhead, fairness and reliability.

2.2 The PDR Protocol

To begin, it is assumed that there is a monitor station which will initialize the ring and designate S ($S > 1$) equally spaced empty tokens circulating the ring.

Figure 2.1 shows the token format used to start a packet transmission

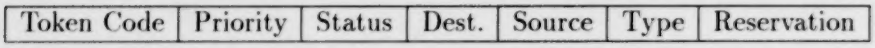


Figure 2.1: Token format for Start transmission

Figure 2.2: Token format for all other states

and Figure 2.2 is the format used for all other cases. The only difference between the two lies in the inclusion of the source and destination addresses when starting a packet transmission.

- **Token Code** is a unique pattern of bits to let the nodes identify the start of a token.

- **Status** is a 3-bit field that contains information controlling access to the ring. Table 2.1 gives the eight possible states of a token indicated by the **Status** bits. Each of the three bits indicates a particular state of the token:

    1. The first bit is the **Start/Continuation (S/C)** bit and indicates if the token holds the start of a packet transmission or the continuation of a transmission (only a **Start** transmission contains the source and destination addresses).

    2. The second bit is the **End/NotEnd (E/N)** bit and indicates if

the token holds the end of the packet transmission.

3. The third bit is the **Source/Destination (S/D)** bit and indicates whether the source or the destination node is responsible for releasing the token.

- The 1-bit field **Type** is used to indicate if the token is a **Cut** token or a **Normal** token. A **Normal** token can be in two possible states: 1) the token is empty, or 2) the token has been used but not yet released[1]. A **Cut** token is formed when a **Normal** token has been used and then released. It is set to **Cut** by the source node if there is sufficient bandwidth for the source node to append an empty token at the end of its packet transmission. Otherwise the token is set to **Cut** by the destination node upon reaching its destination.

- The fields **Priority (P)** and **Reservation (PR)** are used to implement priority and fairness in the transmission of packets. The use of these fields in priority handling will be explained later in Chapter 4 and their use in implementing fairness will be described in Chapter 2.4.

Referring to Table 2.1, note that the **Empty** and **Reserved** states are special cases. A **Reserved** state means that the token is being used for a continued transmission but there is no data attached to it. A detailed expla-

[1]To release a token is to insert an empty token into the ring.

Status bits			State of Token	Abbrev.
S/C	N/E	S/D		
0	0	0	Empty	-
0	0	1	Start packet, NotEnd packet, Dest. rel.	SND
0	1	0	Start packet, End packet, Source rel.	SES
0	1	1	Start packet, End packet, Dest. rel.	SED
1	0	0	Cont. packet, NotEnd packet, Source rel.	Reserved
1	0	1	Cont. packet, NotEnd packet, Dest. rel.	CND
1	1	0	Cont. packet, End packet, Source rel.	CES
1	1	1	Cont. packet, End packet, Dest. rel.	CED

Table 2.1: The 8 different states of a token indicated by the status field

nation on the use of this **Reserved** state will be given in the description of the Transmitter Process in Chapter 2.2.3. An **Empty** state simply indicates an **Empty** token.

2.2.1 Node Table

Every node has a special table that is called the *Node Table* which is maintained in a distributed manner. Figure 2.3 shows the *Node Table* for node i. Let N be the total number of nodes connected to the ring and let each node be assigned an Id (address) sequentially in the direction of data flow around the ring. The *Node Table* contains N entries where the first entry is the node maintaining the table followed by the next downstream node and so on. The 1-bit **Active/Inactive** field indicates if the node is currently transmitting a packet and the **Destination** field indicates the destination address of the packet being transmitted. The last field **Priority**, is used in

Active/Inactive	Node Id	Destination	Priority
	i		
	i+1		
	.		
	.		
	0		
	N-1		
	.		
	.		
	i-1		

Figure 2.3: Node Table

the implementation of priority transmissions and will be explained later in Chapter 4. It is assumed that when the ring is initialized, all entries in the *Node Table* will be set to **Inactive**.

2.2.2 Token Table

Every node is also equipped with a second table called the *Token Table* as shown in Figure 2.4. Let S be the maximum number of tokens allowed to circulate in the ring. The tokens are numbered sequentially around the ring and the entries in the *Token Table* are sorted by the Token Id. Note that the numbering of the tokens is implicit since there is no *token number* field in the token format and also the nodes do not number the tokens in the same way.

Active/Inactive	Token Id	Release	Expected Location	Priority
	0 1 . . S-1			

Figure 2.4: Token Table

- The **Active/Inactive** field in this table indicates if the token is currently present in the ring. This is to facilitate the changing of the number of tokens circulating in the ring and is not needed if the token number is fixed. The dynamic changing of token number will be discussed in Chapter 5 and the **Active/Inactive** field will be temporarily ignored in this chapter as the protocol is described.

- The **Release** field stores the Id of the node that last releases the token.

- The **Expected Location** field gives the estimated location of the token on the ring. Everytime a token visits a node, this field is updated to estimate the location of the token on its next visit. If the token status is **Empty** or **Reserved**, then the token is expected to return after one round trip delay, R. If the token is full, then it is expected to return after time $R + l$ where l is the length of the token.

- The last field in the table, **Priority**, contains a pointer to a stack which keeps track of the priority levels of the token. The use of this field will be discussed later in Chapter 4.

To transmit packets, receive packets and maintain the tables, each node requires the following three processes: the Transmitter process, the Receiver process and the Table management process. Before describing the different node processes, we define the associated bandwidth of token i as the number of bits that can be transmitted from the actual location of token i to the expected location of the next token (not including the number of bits taken up by token i itself).

2.2.3 Transmitter Process

Any node that wants to transmit, has to first capture an **Empty** token. When an **Empty** token arrives, the node needs to know the associated bandwidth of the token (which can be read from the Token Table) to decide if it can use the token and also which status and type bits to set.

Setting the first status bit (**S/C**)

The setting of the **S/C** bit to **Start** serves to inform all nodes that a new packet transmission has begun and the entry corresponding to the source node in the *Node Table* is to be set to **Active**. To start a packet transmission, there must be enough bandwidth to contain both the source

and destination addresses. If there is enough bandwidth, then the **S/C** bit will be set to **Start** and transmission can begin. Since there is more than one token in the ring, a second token might arrive before the packet is fully transmitted. If so, the node will stop the transmission temporarily and look for another empty token. The node must use the next empty token that comes along regardless of the associated bandwidth. If the next empty token has an associated bandwidth of less than one token length, the token status is marked as **Reserved** and no data is transmitted. Otherwise, the **S/C** bit is marked as **Continuation** and the packet transmission is resumed.

Setting the second status bit (**E/N**)

The setting of the **E/N** bit to **End** is to inform all nodes that the packet transmission has been completed and that the Node table is to be updated accordingly. So if there is enough bandwidth to complete the packet transmission, the **E/N** bit is set to **End**, otherwise it is set to **NotEnd**.

Setting the third status bit (**S/D**)

The **S/D** bit is used to indicate whether the source or the destination node is responsible for releasing the token so that the appropriate address can be entered into the **Release** field of the *Token Table*. If there is enough associated bandwidth to complete the packet transmission and also append an empty token then the **S/D** bit is set to **Source** and the token is released right after the end of the packet transmission. Otherwise the responsibility of releasing the empty token is passed on to the destination node, that is by

setting the **S/D** bit to **Destination**.

Setting the **Reserved** state

If two tokens are travelling closely together such that the associated bandwidth of the first token is less than one token length, then the first token cannot be used for any data transmission because there is insufficient space to release the token. However in a continued transmission, the source node is expected to use any empty token that arrives. Therefore, the source node still has to capture the token even if it has insufficient bandwidth and mark it as **Reserved**. Otherwise the token might be used by the next active node downstream to continue its transmission and other nodes will incorrectly assume that the token is being used by the first active node. Hence the purpose of the **Reserved** token is to maintain the correct sequence of nodes using the token so that the tables at all nodes can be updated correctly. The token status will be set back to **Empty** by the source node after it has propagated one complete round.

Setting the **Type** bit

The **Type** bit is used to indicate if a token has been released. It is set to **Cut** by the source node only if there is enough room for the source node to release the token at the end of its packet transmission. Otherwise the **Type** bit remains unchanged to indicate a **Normal** token. The algorithm for the Transmitter Process is presented in Figure 2.5.

Removal of data and token headers

```
Step 1.  Wait for arrival of empty token

Step 2.  If (Start of packet transmission) then begin
             If (associated bandwidth can contain the address fields) then
                 set S/C bit to Start
             Else
                 goto Step 1
         end
         Else (Continuation of packet transmission) begin
             If (associated bandwidth is less than one token length) then begin
                 set token status to Reserved
                 goto Step 1
             end
         end

Step 3.  If (associated bandwidth can contain entire packet) then
             set E/N bit to End
         Else begin
             set E/N bit to NotEnd
             set S/D bit to Destination
             transmit data until expected location of next token
             goto Step 1
         end

Step 4.  If (enough bandwidth to transmit packet and release token) then begin
             set S/D bit to Source
             set Token Type to Cut
             transmit entire packet
             append empty token at the end of packet transmission
         end
         Else begin
             set S/D bit to Destination
             transmit entire packet
         end

Step 5.  End.
```

Figure 2.5: Algorithm for Transmitter Process

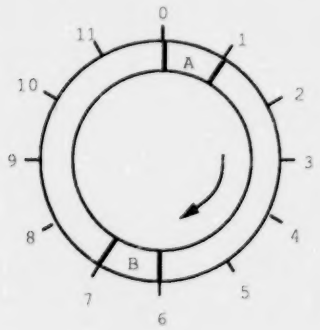
When a token reaches its destination, the data appended to the token header will be removed by the destination node, but the token header continues to propagate around the ring in the form of a **Cut** token. The **Cut** token will then be removed by the source node on its return. If the **S/D** bit is set to **Destination**, then the destination node is responsible for setting the token type to **Cut**, otherwise the token type would have already been set to **Cut** by the source node. The only instance a **Cut** token is not formed is when the token status is **Reserved**.

For the Receiver Process to identify the end of the data stream transmitted, the Transmitter Process must append an "*End-Of-Data*" (EOD) token. For source released transmissions, the empty token released at the end of a packet transmission automatically becomes the EOD token. However for destination released transmissions, there are two situations whereby inserting the EOD token is a problem.

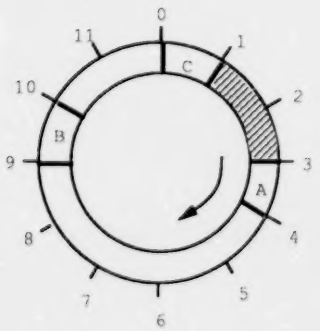
The first situation arises when the token status is **SED** or **CED**, that is the source node has enough bandwidth to complete its packet tranmission but not enough to append an empty token. Hence there is a small gap between the end of transmission and the arrival of the next token. The length of this gap will be less than one token length. The solution to this problem is based on the method proposed by [15] where a token code can be stretched if it is of the form 0111 1110. The source node will have to stretch the length of the arriving token by inserting a zero right after its packet transmission has

completed. Then it will continue to insert a string of one's until the next token arrives. Hence the stretched token code will be of the form $0111\cdots1110$. At the destination node, the Receiver Process is able to set the token code back to its original length since it knows the expected location of the token.

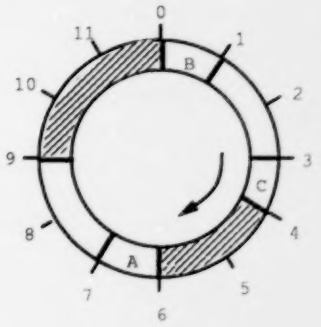
The second situation is illustrated with the following example. Consider a network consisting of 12 nodes and 2 tokens with data flowing in a clockwise direction. The notation "$1E$" indicates that token 1 is empty and the notation "$2F(1 \rightarrow 2)$" indicates that token 2 is being used to transmit data from node 1 to node 2. Let the total ring delay be equal to R. The shaded areas represent data being transmitted.



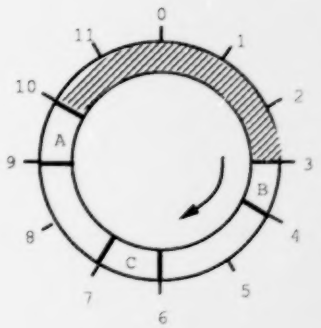
At time $T = 0$:
$[A]$: $1F(0 \rightarrow 6)$, type=Cut.
$[B]$: $2E$.
Node 0 begins its packet transmission.



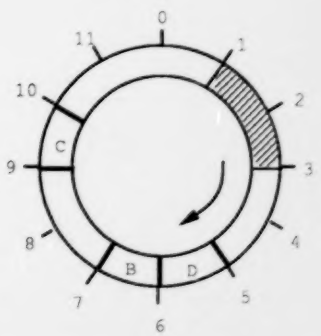
At time $T = \frac{1}{4}R$:
$[A]$: $1F(0 \rightarrow 6)$, type=Cut.
$[B]$: $2F(9 \rightarrow 3)$, type=Normal.
$[C]$: $1E$.
Node 9 begins its packet transmission.
Node 0 completes its transmission and releases the token.



At time $T = \frac{1}{2}R$:
$[A]$: $1F(0 \rightarrow 6)$, type=Cut.
$[B]$: $2F(9 \rightarrow 3)$, type=Normal.
$[C]$: $1E$.
Node 6 begins data reception.
Node 9 is still transmitting.



At time $T = \frac{3}{4}R$:
$[A]$: $1F(0 \rightarrow 6)$, type=Cut.
$[B]$: $2F(9 \rightarrow 3)$, type=Cut.
$[C]$: $1E$.
Node 9 stops transmission temporarily.
Node 3 sets token $[B]$ to Cut, begins data reception and releases empty token behind $[B]$.

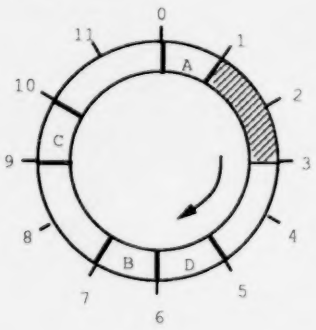


At time $T = R$:
$[B]$: $2F(9 \rightarrow 3)$, type=Cut.
$[C]$: $1F(9 \rightarrow 3)$, type=Normal.
$[D]$: $1E$.
Node 9 continues its transmission using token $[C]$.
Token $[A]$ was removed by the source node 0.

Notice the gap between the end of data transmission and token $[C]$, this gap will be considered as part of the data by the destination node 3 because there is no EOD token. The solution to the problem is as follows:

The Transmitter Process of the source node will be responsible for re-

moving a returning **Cut** token only if it is not preceded by a data stream. Otherwise the **Cut** token will continue to propagate and subsequently be removed by the destination node of the token preceding it. The **Cut** token is not removed by the source node so that it can be used as the (EOD) token for the data stream before it. The following diagram shows the result of this change.



At time $T = R$:
$[A]$: $1F(0 \rightarrow 6)$, type=Cut.
$[B]$: $2F(9 \rightarrow 3)$, type=Cut.
$[C]$: $1F(9 \rightarrow 3)$, type=Normal.
$[D]$: $1E$.
Token $[A]$ not removed by its source node 0.
It will be removed later by node 3,
destination of the preceding token.

2.2.4 Receiver Process

Each node can transmit one packet to one destination node at a time, but since more than one node is transmitting at the same time, a destination node may be receiving data that belong to different sources. To correctly reassemble the packets, the destination node needs to know the source of the data. The question that each node should resolve is what the source and destination addresses of a continuation transmission are. By allowing a used token to propagate back to its source after arriving at its destination in the form of a **Cut** token, all nodes are informed of any packet transmission

```
Step 1.  If (source = i) and (token status = Reserved) then begin
             Set token status to Empty.
             Goto step 3.
         End
Step 2.  If (destination = i) and (token status ≠ Reserved) then begin
             If (S/D bit = Source) then begin
                 Receive data until a token is encountered.
                 Goto step 3.
             End
             Else If (S/D bit = Destination) then begin
                 Set token type to Cut.
                 Append an Empty token to end of Cut token.
                 Receive data until a token is encountered.
             End
         End
Step 3.  End.
```

Figure 2.6: Algorithm for Receiver Process

that has taken place. This enables the nodes to keep the information in the tables current, which is used to find the source and destination addresses of a continued packet transmission. The Table Management Process will compute the addresses based on the information in the tables, then pass them to the Receiver Process.

In the algorithm shown in Figure 2.6, it is assumed that when the Receiver Process for node i releases an empty token, the Transmitter Process at the same node may choose to use the empty token if it has a packet to transmit. A node, i, upon receiving a token will perform the steps shown in Figure 2.6.

2.2.5 Table Management Process

Other than transmitting and receiving data, each node also needs to update the two tables everytime a token passes by. The token can either be a **Cut** or a **Normal** token. However a **Cut** token that is received before the expected location of the current token[2] is ignored because it is being used to mark the end of a data stream. Therefore, the Table Management Process will only take into account tokens which are received at or after the expected location of the current token. Note that a *stretched* token [3] may arrive early but is completely received only at or after the expected location.

A **Normal** token may be preceded by several **Cut** tokens as a token can be captured and released several times in one round trip. The same entry in the *Token Table* is updated as long as the token is a **Cut** token. A pointer is used to access the entries in the *Token Table* and the pointer is incremented to access the next entry only after a **Normal** token arrives and updating has been completed. Assuming that when the pointer reaches the entry for token $N-1$ in the table, it is reset to point to the entry for token 0. The *Node Table* is updated only when the token status is **SND, CES** or **CED**. Figure 2.7 shows the algorithm for the Table Management Process which is executed every time a token arrives.

[2]The current token is the token whose entry in the Token Table is currently being accessed by the table pointer.

[3]Refer to page 27 for definition of stretched token.

Step 1. If (S/C bit = **Continuation**) then begin
Get previous node (PN) that released the token from
Release field in Token Table.
Start from entry in Node Table where Node Id = PN,
Do a top-down search for first **Active** entry.
Get source address from the field **Node Id**.
Get destination address from the field **Destination**.
Pass source and destination addresses to Receiver process.
If (E/N bit = **End**) then begin
Search Node Table for entry with Node Id = source address.
Set **Active/Inactive** field to **Inactive**.
end
end

Else (S/C bit = **Start**) begin
Read source and destination addresses from token header.
If (E/N bit = **NotEnd**) then begin.
Search Node Table for entry where Node Id = source address.
Set **Active/Inactive** field to **Active**.
Write destination address to **Destination** field.
end
end

Step 2. If (S/D bit = **Source**) then
Write source address to **Release** field in Token Table.
Else (S/D bit = **Destination**)
Write destination address to **Release** field in Token Table.

Step 3. If (Token Type = **Normal**) then begin
Update **Expected Location** field in Token Table.
Increment Token Table pointer to access next **Active** entry.
end

Step 4. End.

Figure 2.7: Algorithm for Table Management Process

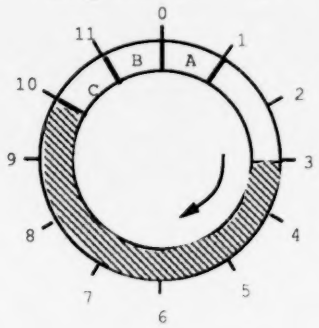
2.2.6 Example

To get a better picture of how the PDR protocol works, the following example illustrates the proceedings of the three processes from the point of view of node 0. The network shown is operating with 12 nodes and 2 tokens, with data propagating in a clockwise direction. The total ring delay is denoted by R and the token length is denoted by l, where l is also equal to the distance between consecutive nodes, therefore $R = 12l$. For the sake of illustration only, it is also assumed that the packet lengths are in multiples of l.

At time $T = 0$

There are 3 active entries in the Node Table and the Token Table pointer is currently accessing the first entry. The state of the ring and the tables are shown on the next page. The shaded area in the ring diagram represents data being transmitted, the *current entry* in the Token Table and the *next active entry* in the Node Table are indicated by bold prints.

Ring State at $T = 0$



Token Table

Token	Rel.	Loc.
1	**4**	**0**
2	11	$2l$

Node Table

A/I	Node	Dest.
I	0	-
	⋮	
A	3	10
A	**4**	**0**
	⋮	
A	9	3
	⋮	

[A]: $1F(2 \rightarrow 4)$, status=SED, type=Cut
[B]: $1F(4 \rightarrow 0)$, status=Reserved, type=Normal
[C]: $2F(3 \rightarrow 10)$, status=CED, type=Cut

Computations performed at node 0 on reception of token [A]:

1. S/C bit=**Start** ⇒ Source and destination addresses are read from the token header, source=2 and destination=4.

2. E/N bit=**End** ⇒ No need to update Node Table since transmission has started and completed.

3. S/D bit=**Destination** ⇒ Update **Release** field in Token Table to 4.

At time $T = l$

Computations performed at node 0 on reception of token [B]:

1. S/C bit=**Continuation** ⇒ The addresses are computed by the Table Management Process. Node 4 is the last node to release the to-

ken(obtained from Release field in Token Table), the next active entry in Node Table is: source=4, destination=0.

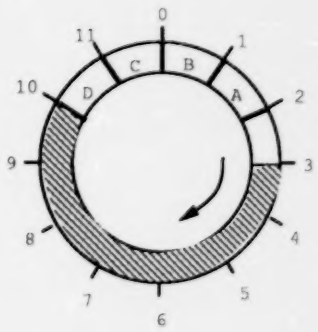
2. S/D bit=**Source** ⇒ Update **Release** field in Token Table to 4.

3. Type=**Normal** ⇒ Update expected location to $R + l$ and increment Token Table pointer.

   (Note that the status of token [B] is the special case **Reserved**, therefore the new location is $R + l$ instead of $R + 2l$.)

The state of the tables after the update is shown next.

Ring State at $T = l$



Token Table

Token	Rel.	Loc.
1	4	$R + l$
2	**11**	$2l$

Node Table

A/I	Node	Dest.
I	0	-
	⋮	
A	3	10
A	**4**	**0**
	⋮	
A	9	3
	⋮	

[*B*]: $1F(4 \rightarrow 0)$, status=Reserved, type=Normal
[*C*]: $2F(3 \rightarrow 10)$, status=CED, type=Cut
[*D*]: $2E$.
[*A*]: $1F(2 \rightarrow 4)$, status=SED, type=Cut

At time $T = 2l$

Computations performed at node 0 on reception of token [C]:

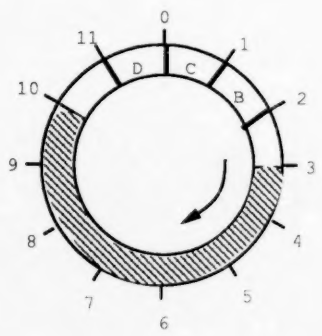
1. S/C bit=**Continuation** ⇒ The addresses are computed by the Table Management Process. Node 11 is the last node to release the token(obtained from Release field in Token Table), the next active entry in Node Table is: source=3, destination=10.

2. E/N bit=**End** ⇒ Set current entry in Node Table to Inactive.

3. S/D bit=**Destination** ⇒ Update **Release** field in Token Table to 10.

Note that token [A] has been removed by its source node 2.

Ring State at $T = 2l$



Token Table

Token	Rel.	Loc.
1	4	$R + l$
2	**10**	$2l$

Node Table

A/I	Node	Dest.
I	0	-
	⋮	
I	**3**	**10**
A	4	0
	⋮	
A	9	3
	⋮	

[C]: $2F(3 \rightarrow 10)$, status=CED, type=Cut
[D]: $2F(11 \rightarrow 2)$, status=SES, type=Cut
[B]: $1F(4 \rightarrow 0)$, status=Reserved, type=Normal

At time $T = 3l$

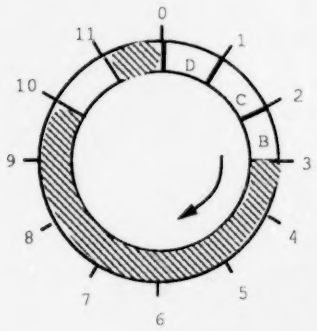
Computations performed at node 0 on reception of token [D]:

1. S/C bit=**Start** ⇒ Source and destination addresses read from the token header are: source=11 and destination=2.

2. E/N bit=**End** ⇒ No need to update Node Table since transmission has started and completed.

3. S/D bit=**Source** ⇒ Update **Release** field in Token Table to 11.

Ring State



Token Table

Token	Rel.	Loc.
1	4	$R + l$
2	**11**	$2l$

Node Table

A/I	Node	Dest.
I	0	-
	⋮	
A	4	0
	⋮	
A	9	3
	⋮	

[D]: $2F(11 \rightarrow 2)$, status=SES, type=Cut
[B]: $1F(4 \rightarrow 0)$, status=Reserved, type=Normal
[C]: $2F(3 \rightarrow 10)$, status=CED, type=Cut

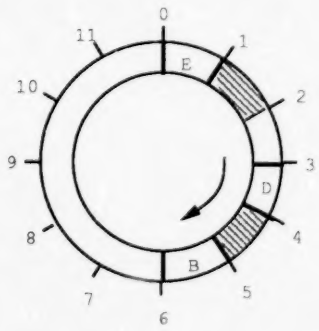
At time $T = \frac{1}{2}R$

Computations performed at node 0 on reception of token [E]:

1. Token status=**Empty** $\Rightarrow$ Update expected location to $R + \frac{1}{2}R$ and increment Token Table pointer.

Note in the following diagram that token [C] has been removed by its source node 3.

Ring State at $T = \frac{1}{2}R$



Token Table

Token	Rel.	Loc.
1	**4**	$R + l$
2	11	$R + \frac{1}{2}R$

Node Table

A/I	Node	Dest.
I	0	-
	⋮	
A	4	0
	⋮	
A	9	3
	⋮	

[E]: $2E$.
[B]: $1F(4 \rightarrow 0)$, status=CED, type=Normal
[D]: $2F(11 \rightarrow 2)$, status=SES, type=Cut

At time $T = R + l$

Computations performed at node 0 on reception of token [B]:

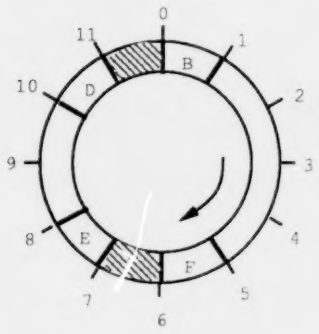
1. S/C bit=**Continuation** $\Rightarrow$ The addresses are computed by the Table Management Process. Node 4 is the last node to release the token(obtained from Release field in Token Table), the next active entry in Node Table is: source=4, destination=0.

2. E/N bit=**End** $\Rightarrow$ Set current entry in Node Table to **Inactive**.

3. S/D bit=**Destination** $\Rightarrow$ Update **Release** field in Token Table to 0.

4. Type=**Normal** $\Rightarrow$ Update expected location to $2R+2l$, set token Type to **Cut**, append empty token $[G]$ and increment Token Table pointer.

Ring State at $T = R + l$



Token Table

Token	Rel.	Loc.
1	0	$2R + 2l$
2	**11**	$R + \frac{1}{2}R$

Node Table

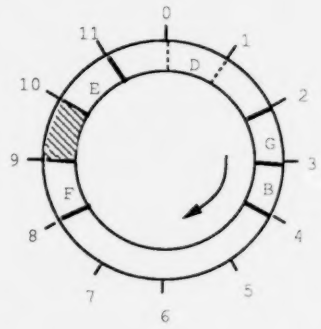
A/I	Node	Dest.
I	0	-
	⋮	
I	**4**	**0**
	⋮	
A	9	3
	⋮	

$[B]$: $1F(4 \rightarrow 0)$, status=CED, type=Normal$\rightarrow$Cut
$[D]$: $2F(11 \rightarrow 3)$, status=SES, type=Cut
$[E]$: $2F(5 \rightarrow 0)$, status=SES, type=Cut
$[F]$: $2E$.

At time $T = R + 4l$

Note that token [D] was not removed by its source node 11 because it is used as an EOD indicator and therefore arrives early at node 0. Node 0 expects the next token to arrive at time $T = R + \frac{1}{2}R$, hence token [D] is removed by node 0 together with the data stream.

Ring State at $T = R + 4l$



Token Table

Token	Rel.	Loc.
1	0	$2R + 2l$
2	**11**	$R + \frac{1}{2}R$

Node Table

A/I	Node	Dest.
I	0	-
	⋮	
A	9	3
	⋮	

[D]: Removed by node 0.
[E]: $2F(5 \rightarrow 0)$, status=SES, type=Cut
[F]: $2E$.
[B]: $1F(4 \rightarrow 0)$, status=CED, type=Cut
[G]: $2E$.

2.3 Overhead

The overhead incurred per token is three bits for the **Status** and one bit for the **Type**, plus the length of the token code. For every packet sent, there is also the overhead for the source and destination addresses. Compared to the overhead incurred in the IEEE 802.5 protocol [10], the new protocol only requires an additional three bits of overhead, not to mention the difference between the total ring delay, R and the frame length, l, that is incurred in the IEEE 802.5 when l is less than R.

The overhead per packet is dependent on how many tokens are used to transmit the packet. Having more tokens circulating in the ring, results in lesser bandwidth associated with each token and subsequently a higher number of tokens being required to transmit a packet. Therefore, given a certain load level, increasing the number of tokens in the ring beyond a certain threshold will cause the overhead incurred to overwhelm any reduction in access delay. In Chapter 3, a simulated model of the protocol is used to examine this threshold.

In a ring network, a node receives data traffic from its upstream node then retransmits the traffic to its downstream node, hence a minimum delay of one bit is required at each node. For a source release token ring protocol like FDDI [23], only a 1-bit delay is needed to check if the token is empty or full. For a destination release protocol like the Orwell ring [1], a delay equal to

the length of the destination address is required because the destination node must be determined before a token can be released. In the new protocol, the destination node only removes the data but not the token header, therefore the delay at each node is only three bits, which is the time needed to read and update the token status.

The highest amount of computation that needs to be done is when a **Continuation** token is received. That is why the **S/C** bit has been placed as the first bit in the token status so that computation of the source and destination addresses can begin right after reading the first bit of the token status.

In the *Token Table*, the locations of all tokens are recorded in the **Expected Location** field and a token will arrive at or after the expected location. A **Cut** token that arrives before the expected location is simply the *End-of-data Marker* and is ignored. This approach prevents collisions but at the expense of some bandwidth waste whenever a token arrives late. The amount of wasted bandwidth depends on the number of times a token is captured and released within one round trip delay.

2.4 Fairness

Fairness in a network is defined in [4] as "*The provision of approximately the same possibility of access to the medium and the same level of service, to all*

the stations". For source release protocols, all nodes receive a fair distribution of the bandwidth as tokens are released and passed sequentially around the ring. For destination release protocols, this inherent fairness mechanism may be nonexistent. The distribution of bandwidth now depends on a few factors:

- The distribution of the lengths of packets transmitted.
- The number of nodes between the source and destination of the packets transmitted.
- The location of active source nodes with respect to the nodes responsible for token release.

Different mechanisms have been used to solve this fairness problem in destination release protocols and the following is a survey of these mechanisms.

2.4.1 DSDR protocol

In [4], the DSDR (Destination-Stripping Dual Ring) protocol was defined for a slotted medium, dual ring network where slots are released by their destinations. In a dual ring network, there are two rings with traffic propagating in opposite directions. The two rings are used symmetrically by each station and the choice of which ring to send a packet on depends on the destination address (the shorter of the two paths will be chosen). The rings operate independently and the following descriptions refer to only one of the rings.

The DSDR protocol uses a token to enforce fairness in the distribution of bandwidth. The meaning of the token here is different from that in a token ring protocol like FDDI. It is only used for the purpose of enforcing fairness. The node holding the token is responsible for generating a predetermined fixed number of slots. Let this number be *Max*. The node can fill these slots if it has packets to send or just leave them empty. When a slot is generated, it has a counter field which is initialized to a value equal to the distance (in nodes) between the current node to the next token holder (assuming the token has already been passed to some other node).

After generating the *Max* number of slots, it then passes the token to the next node (token passing can be either implicit or explicit). In this manner, the node holding the token receives top priority to fill a *Max* number of slots. This is repeated with all nodes in a round robin fashion. Nodes without the token can also transmit if they encounter empty slots and if the number of nodes between the source and the destination is less than the value of the slot counter (i.e., the destination of the packet is before the node holding the token). The maximum number of slots that each node is permitted to generate also sets an upper bound on the access delay. The value of *Max* can thus be computed to satisfy the required upper bound.

2.4.2 Fairness mechanism in Counter Rotating Rings

In [5], two types of fairness mechanisms were defined for a full-duplex[4] destination release buffer insertion ring: the global fairness mechanism, and the local fairness mechanism. Access control to a buffer insertion ring is distributed. That is, control information is not obtained from the ring; a node can start transmission as long as its insertion buffer is empty. If data traffic from upstream arrives during the transmission of a packet, this data traffic is temporarily stored in the insertion buffer until the packet transmission is complete.

Global fairness mechanism

The global fairness mechanism is implemented with the use of control messages called SATs which are passed around the ring in a direction opposite to data flow. Each node is allowed to transmit a maximum number of packets *Max* and a counter is used to keep track of the number of packets already transmitted. When a node receives the control message SAT, it resets the counter back to zero. Hence the value *Max* is the maximum number of packets transmitted by a node between two successively received SAT messages. A node will hold the SAT message if it has packets to transmit and the counter is less than *Max*. Otherwise it will pass the SAT message upstream.

[4]Data can propagate in opposite directions simultaneously.

By holding the SAT message, a node is preventing the upstream node from resetting its counter after it reaches *Max*, hence leaving the bandwidth free for the downstream node to transmit. When a SAT message arrives at a node that is already holding another SAT message, the two SAT messages are merged into one. When a SAT message is lost, another is generated after a time-out. The *Max* number of packets transmitted between SAT messages sets an upper bound on the access delay.

The global fairness mechanism enforces fairness among all nodes even when starvation occurs within a subset of nodes which communicate only within the subset. It also operates at all times even when there are no starved nodes. The following fairness mechanism specifically addressses these two problems.

Local fairness mechanism

The local fairness mechanism recognizes two access modes and is implemented by using two control messages. The two access modes are defined as restricted and unrestricted. A node in the unrestricted mode transmits packets as in the buffer-insertion protocol, while a node in restricted mode is allowed to transmit only one packet. The control message REQ sets a node into restricted mode and the message GNT resets the node into unrestricted mode. The control messages are transmitted to the upstream node in the opposite direction of data traffic. All nodes operate in the unrestricted mode

until some node is starved. The starved node will send the message REQ message upstream and set the upstream node into restricted mode. If the upstream node has a full insertion buffer, it will in turn send another REQ upstream. Thus a chain of nodes will be set into restricted mode until a node with an empty insertion buffer is reached. The node that initiated the chain of REQ messages will be responsible for sending the GNT control message when it has transmitted its packets. Again this sets a chain of GNT messages upstream until the nodes are back in the unrestricted mode. Therefore, the local fairness mechanism is activated only when starvation occurs and its operation is restricted to the subset of nodes involved.

2.4.3 Orwell protocol

In [1], the Orwell protocol and its fairness mechanism were described for a high-speed slotted ring medium. The protocol recognises two service classes: 1) class 1 for delay-sensitive data that requires guaranteed bandwidth and 2) class 2 for delay-tolerant data. The fairness mechanism was defined only for class 1 data. To implement the mechanism, each node has a counter that keeps track of the number of packets transmitted. When the counter reaches the maximum number (Max) of packets allowed, the node goes into the *Pause* state where it is temporarily inhibited from transmitting any packets until it receives the control message RESET. Once the RESET message is received,

the counter is reset back to zero and the node gets out of the *Pause* state. While a node is in the *Pause* state, it monitors activity on the ring by setting the *Trial* bit of a passing empty slot to ON. If the empty slot is used, the *Trial* bit is set OFF and the trial fails. The trial is successful when the empty slot returns unused, and then the node can send a RESET message alerting all nodes of the successful trial. Any node in the *Pause* state can start a trial and send the RESET message. Since each node is permitted to transmit no more than *Max* number of packets between RESET messages, it sets an upper bound on the access delay.

2.4.4 PDR protocol

After reviewing the various fairness mechanisms that have been proposed, we will proceed to define a fairness mechanism for the PDR protocol. Except for the Orwell protocol fairness mechanism, the rest of the mechanisms that we have reviewed are defined for dual counter-rotating rings where control information is sent in a direction opposite to data flow. Since the PDR protocol is defined for a high-speed token ring network with data propagating only in one direction, a fairness mechanism similar to that used by the Orwell protocol is implemented. The only difference between the two mechanisms lies in the method used for monitoring activity on the ring.

Each node will need a *Counter* to keep track of the number of bits trans-

mitted. The number of bits is used here instead of number of packets because in this protocol, the packet length is not fixed and thus counting the number of bits is a more accurate measure of bandwith utilization. Nodes having transmitted more than a *Max* number of bits will be temporarily disallowed to capture the next empty token until a RESET control message is detected. The RESET control message used here can simply be a special code in the **Reservation** field of the token header.

A node waiting for the RESET message will begin to monitor ring activity. The *Trial* bit used in the Orwell protocol is not needed. Activity on the ring is monitored simply by observing any passing empty token with no cut tokens [5] before it. If after propagating once around the ring, the same token is still empty with no cut tokens preceding it, this means the empty token was not used by any nodes on the ring. The first node to observe this can reset its *Counter*, use the empty token and set the **Reservation** field to indicate a RESET message. This same node is also responsible for removing the RESET message after it has made its round. All other nodes detecting the RESET message will also reset their counters.

In Chapter 3, this fairness mechanism will be implemented in the simulated model to investigate its effectiveness.

[5] A cut token is formed when a token has been used and then released.

2.5 Reliability

In a ring network, all nodes are linked together forming a closed loop where each node is connected to its two neighbouring nodes, one upstream node and one downstream node. Data is passed from one node to another until it reaches the destination node. If a single node in the ring fails, the communication chain is broken and the entire ring network fails.

To overcome this reliability problem, variations to the ring topology have been defined, for example : the star-ring topology [24], dual counter-rotating rings [23] and the chordal ring [2]. These variations to the ring topology will not be addressed in this section. Here the main reliability concern is the detection of errors in data transmission and also the recovery of such errors.

It is assumed that a monitor station will be used to initialize the ring, to detect errors and to recover from them. The successful operation of the PDR protocol depends heavily on the information bits in the token header and the tables being error-free. Both the **Normal** and the **Cut** tokens play an important role in ensuring that the tables are correctly updated. A single bit of error in the token status can lead to a breakdown of the protocol.

To check for errors in the token header, a frame check sequence (FCS) can be added to the token format as shown in Figure 2.8. With the FCS, the monitor station can thus detect most errors occuring in the token headers. Depending on the severity of the error, the monitor station will decide if the

Token Code	Priority	Status	Dest.	Source	Type	Res.	FCS

Figure 2.8: Token Format with FCS

ring needs to be reinitialized.

One way to improve the network reliability and also reduce the protocol complexity is to include the source address in the token headers of continuation transmissions. The search procedure in step one of the Table Management Process will not be required as the correct entry in the *Node Table* can be accessed directly using the source address read from the ring. The same search procedure can instead be used as an error check to confirm that the information received is correct. Having the source address in all token headers also makes error detection and recovery easier as the protocol is now much less susceptible to failures due to single bit errors. However overhead would be significantly increased especially when a high number of tokens are used.

The tokens used for transmitting a packet are released before the destination node has completely received the data, therefore it is not possible for the destination node to send acknowledgements in the token headers to indicate that the packet received is complete and correct. One solution to the problem is to modify the protocol to let the last token used for transmitting the packet be released by the source node, after the token has propagated

once around the ring. The destination node will thus be able to piggyback an acknowledgement onto the last token back to the source node. This method will suffer a degradation in performance especially when the mean packet length is less than the average associated bandwidth per token. The protocol would resemble a source release protocol if each packet requires only one token to transmit it. Hence intuitively, this method should be implemented with a large number of tokens to achieve a better performance.

Chapter 3

Study of New Protocol through Simulation Model

3.1 Introduction

The PDR protocol proposed in this thesis can be modeled by a *multiserver multiqueue system* (MSMQ) where the servers in the system represent tokens operating in the ring and customers in the queues represent packet arrivals at each node. A few recent analyses of such systems can be found in [18], [21], [13], [3] and [16], but all of them are just approximate analytical models. The MSMQ system is also referred to as a *multiserver polling model* because the queues are served cyclically by the servers which is an extension of the single server polling system. In the PDR protocol, tokens can be released by both

the source and destination nodes, hence the queues are not served cyclically by the servers. Therefore deriving even an approximate model of the PDR protocol will be extremely difficult. Instead, we will study the performance of the protocol through a simulation model. In the following sections, we describe in detail the simulation model and the performance measure used to study the protocol. Different number of tokens are used to examine its effect on protocol performance. The PDR protocol is then compared to other protocols like FDDI, the conventional slotted ring, DQDB and the source-release multi-token ring protocol by [11]. The effectiveness of the fairness mechanism proposed in Chapter 2.4 and the amount of performance degradation incurred by the implementation is also examined through the simulation model. A brief discussion is then presented on the upper bounds on access delay with the implementation of the fairness mechanism.

3.2 Simulation Model

To study the performance of the Partial-Destination-Release Multi-Token (PDR) protocol, a simulator was written to model the protocol based on the following parameters and assumptions.

Parameters:

- The ring consists of 50 nodes operating at a speed of 100 Mb/s.

- The total ring delay is 1000 bits including the 3 bits of delay at each node. This is about equivalent to a 2 kilometre long ring.
- Packet interarrival times are exponentially distributed.
- Packet lengths are exponentially distributed with a mean of 200 bits.
- Length of the token header is 10 bits.
- Overhead for the source and destination addresses is 12 bits.

Assumptions:

- The nodes are equally spaced around the ring.
- The destination of a packet is uniformly distributed among the nodes.
- Each node has an infinite number of output buffers to store packets awaiting transmission.
- Each node's receiver station has a number of input buffers equal to $N-1$, where N is the total number of nodes on the ring.

The last assumption is required since multiple nodes may be transmitting packets to the same destination node simulataneously. This is possible because a node may have started a packet transmission but is awaiting more empty tokens to complete the transmission. In the meantime, the destination node must store the subpackets from the different source nodes separately until the rest of the subpackets arrive.

3.3 Performance Measure

The sole performance measure used here will be the mean packet transfer delay which is defined as the time starting from the arrival of a packet at a node until the time the packet is completely received at its destination node. The mean transfer delay can be broken down into three parts:

1. Queuing delay: time spent by a packet waiting in a queue since its arrival until it reaches the head of the queue.

2. Access delay: time spent by the packet at the head of a queue waiting for the arrival of an empty token.

3. Propagation delay: time taken to completely transmit a packet and propagate to its destination node.

The mean propagation delay remains constant since the destination of a packet is uniformly distributed and the packet length is exponentially distributed with a mean of 200 bits. The mean queuing delay is essentially dependent on the mean access delay, so our main concern will be to reduce the mean access delay.

Figure 3.1 shows the performance of a ring operating with 1, 2, 4, 8, 12 and 16 tokens. As the number of tokens operating in the ring is increased, the performance of the protocol improves, i.e. the mean transfer delay is decreased. However this performance improvement diminishes as the number

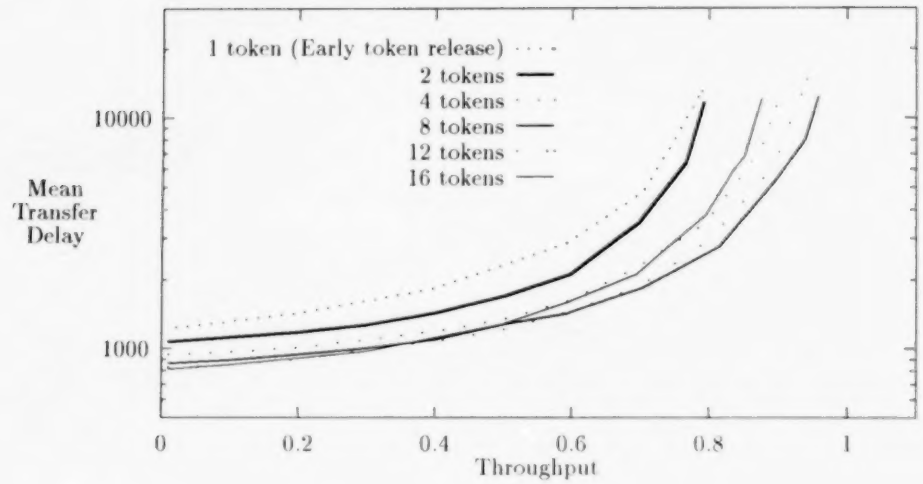


Figure 3.1: PDR protocol with 1, 2, 4, 8, 12 and 16 tokens

of tokens gets larger. At very light load, with tokens equally spaced on the ring, the mean access delay will be close to $\frac{R}{2l}$ where R is the total ring delay and l is the number of tokens operating in the ring. At light load, the mean access delay varies inversely proportional to the value of l, so having more tokens in the ring will reduce the access time. However when the value of l is large, adding more tokens to the ring will have a lesser impact on the reduction in access delay, as can be seen from Figure 3.1.

If the number of tokens is increased beyond a certain threshold, the performance of the protocol shows no significant improvement and starts to degrade at higher loads. After this threshold, the greater the number of tokens, the faster the performance degradation as the load increases. An

important factor that affects the performance of the protocol is the overhead incurred which is dependent upon the number of tokens in the ring. The presence of more tokens will introduce more overhead and will subsequently increase delay in the following ways:

- First of all, the average associated bandwidth for each token is reduced which results in packets having to be broken up into more subpackets. For every subpacket transmitted, the node has to use one empty token. Obviously having more subpackets means having to wait for more empty tokens and also incurring more overhead.

- To start a packet transmission, a node has to look for an empty token with sufficient bandwidth to hold both the source and destination addresses. An empty token with insufficient bandwidth is ignored. When the average bandwidth associated with each token is reduced, the probability of finding an empty token with enough bandwidth for the addresses is also reduced, thereby increasing the access delay. This phenomenon is more pronounced at heavy load.

- With a large number of tokens in the ring, there is a high probability of tokens clustering together, therefore increasing the waiting time.

  Clustering occurs when two or more tokens travel closely together and practically become a single token. For example, if two tokens are clustered together such that the first token has an associated bandwidth

of less than one token length, then the token cannot be used to start a transmission nor send data in a continued transmission. So effectively only one token is operating in the ring and the access delay is therefore increased.

- Finally, the tokens themselves take up bandwidth thus reducing the total amount of bandwidth available for data transmission.

The amount of overhead incurred increases rapidly with increasing the number of tokens and increasing load. This explains why in Figure 3.1 we see a rapid performance degradation as the load increases for the case when 16 tokens are used.

3.4 Comparison to Other Protocols

For a multi-token source release protocol [11], the maximum achievable throughput can be given by the formula

$$\frac{l[\frac{R}{l} - t]}{R + \frac{R}{N}} \times \frac{m}{m + h} \tag{3.1}$$

where

- R = Total ring delay
- t = Length of token
- l = Number of tokens

- m = Mean packet length
- h = Packet overhead
- $\frac{R}{N}$ = Walk time or distance between nodes
- $\frac{R}{l}$ = Average bandwidth per token

based on the assumptions that $(m+h)$ is a multiple of $\frac{R}{l}$ and that the tokens are equally spaced around the ring. The first part of the formula represents the proportion of bandwidth that is available for data transmission out of the total available bandwidth and the second part of the formula represents the proportion of actual data transmitted out of the total transmitted bits which includes the packet overhead.

Based on the assumption that the destination of a packet is uniformly distributed, on the average a packet will have to propagate half way around the ring to reach its destination. For a destination release protocol, it is thus possible to use the remaining half of the bandwidth to transmit more data. If we assume that the PDR protocol is entirely destination release, then the maximum achievable throughput can be defined as

$$(\frac{l[\frac{R}{l}-t]}{R}+\frac{l[\frac{R}{l}-2t]}{R})\times(\frac{m}{m+h}) \tag{3.2}$$

This value is not equal to twice the achievable throughput of the source release protocol because only the data is removed at its destination while the token header continues to propagate back to its source. The first part

of the equation $\frac{l[\frac{R}{l}-t]}{R}$, represents the first half of the bandwidth used for data transmission. The second part of the equation $\frac{l[\frac{R}{l}-2t]}{R}$, represents the remaining half of the bandwidth available for transmission, which takes into account the space occupied by the token headers returning to their source nodes. Note that in this equation, the walk time is equal to zero since a destination node can release and use a token immediately.

The PDR protocol is only partially destination release, hence to compute the maximum achievable throughput, we need to know the proportion of tokens released at their destinations over the total number of tokens released. If we let this value be p, then the formula can be rewritten as

$$(\frac{l[\frac{R}{l}-t]}{R}+\frac{p\times l[\frac{R}{l}-2t]}{R})\times(\frac{m}{m+h}) \tag{3.3}$$

Note that both Equations 3.2 and 3.3 are just approximations.

Having computed the maximum achievable throughput for both the source release protocol and the PDR protocol, we look at Figure 3.2 which compares the performance of these two protocols with 2 and 8 tokens. The PDR protocol outperforms the source release protocol in both cases. This gain in performance seems to be more significant at higher loads and with a larger number of tokens. Another observation is that in the source release protocol, the ring operating with 2 tokens begins to outperform that using 8 tokens when the load is high. However, this is not true for the PDR protocol.

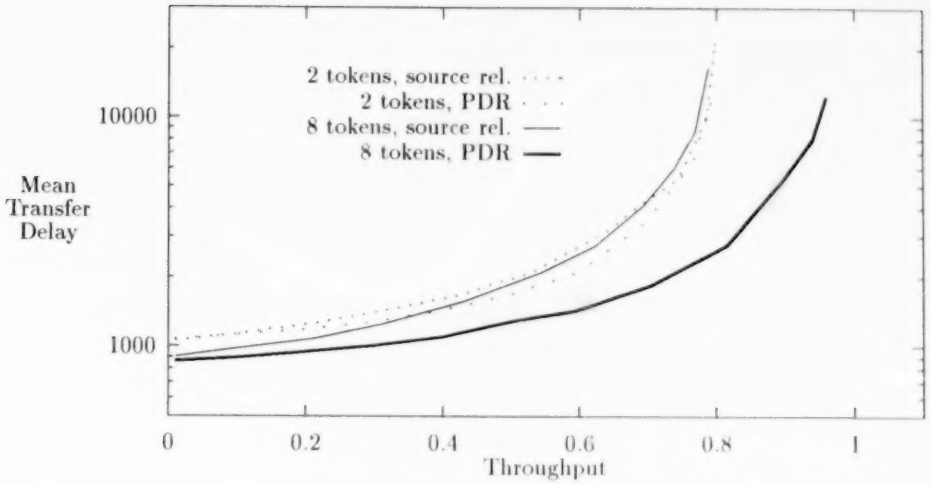


Figure 3.2: Source and Destination release protocols

To understand this difference in behaviour, we refer to Equation 3.3 defined for the maximum achievable throughput of the PDR protocol. A high throughput can be achieved when the value of p is high, which means that more tokens are released at their destination nodes. A token is released at its source node only if there is enough bandwidth to append an empty token at the end of packet transmission. By increasing the number of tokens in the ring, the associated bandwidth per token decreases, therefore the proportion of tokens released by their destinations, i.e. p, increases.

The destination release feature of the PDR protocol provides a gain in throughput that compensates the overhead resulting from the use of more tokens. This explains the difference in performance behaviour between the

PDR and the source release protocols for rings operating with 2 and 8 tokens at high loads. However, further increases in the token number, for example to 16 tokens as in Figure 3.1, will still result in a rapid performance degradation with increasing load as the overhead incurred overshadows the gain in throughput.

Next we compare the performance of the PDR protocol with the conventional slotted ring [9] and the Distributed Queue Dual Bus (DQDB) [19]. The simulation is based on a network of 50 equally spaced nodes with a total ring delay (R) of 1000 bits (which includes the 3-bits delay at each node for the PDR protocol). If the data transmission rate is 100 megabits per second, then the span of the ring is less than 3 kilometres. The message lengths are exponentially distributed with a mean (m) of 500 bits. The number of tokens/slots in all three protocols is equal to 2. Based on the standard DQDB protocol, a slot length of 424 bits is used for both the slotted ring and DQDB protocols. For a fair comparison, an overhead of 72 bits per slot/token and an overhead of 224 bits per message is used for all three protocols. In Figure 3.3, we can see that the PDR protocol outperforms the slotted ring and the DQDB for the entire throughput range, and especially at high loads.

Figure 3.4 shows the simulation results when the mean packet length is increased to 1000 bits and the total ring delay is increased to 5000 bits. The span of the ring will be slightly less than 15 kilometres. With a larger network, the performance of the DQDB is slightly better than the PDR

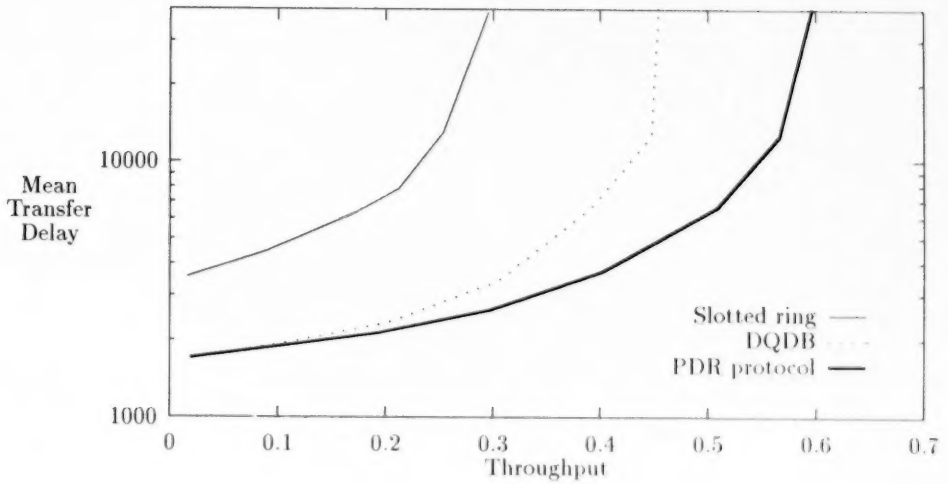


Figure 3.3: Comparison with Slotted ring and DQDB: R=1000, m=500 (exp)

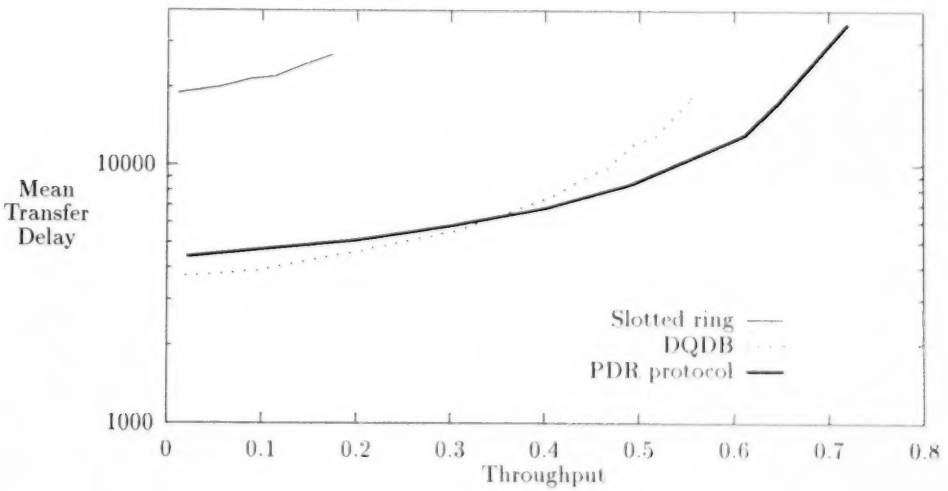


Figure 3.4: Comparison with Slotted ring and DQDB: R=5000, m=1000 (exp)

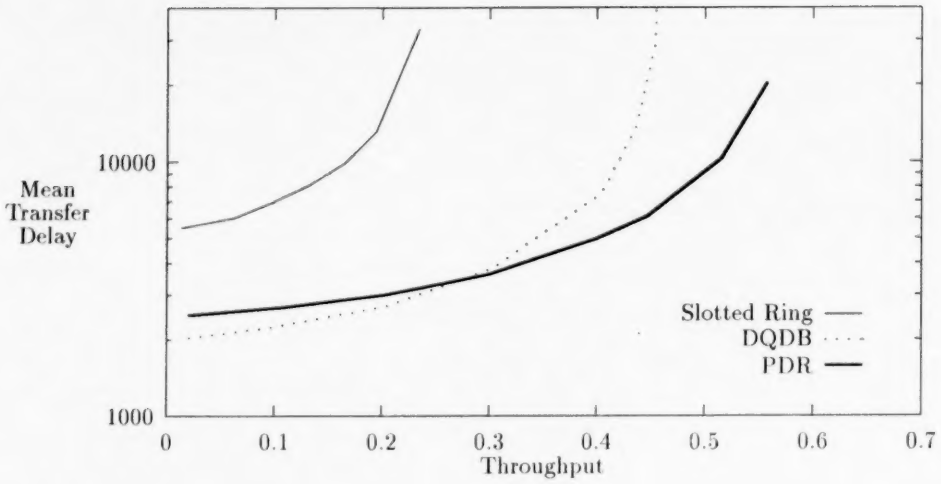


Figure 3.5: Comparison with Slotted ring and DQDB: R=2000, m=500 (exp)

protocol under light load. This is because the mean access delay for DQDB is very close to zero at light loads. However, the PDR protocol quickly overtakes the DQDB as the load increases. The slotted ring on the other hand, does not perform well at all in a large network.

In Figure 3.5, the mean message length is 500 bits and the number of nodes in the ring is increased from 50 to 100. For the previous two cases, the 3-bit delay at the nodes for the PDR protocol is insignificant and therefore not included in the total ring delay. However in this case, the number of nodes in the ring is doubled, therefore increasing the amount of delay at the nodes from 150 to 300 bits. This extra delay of 300 bits for the PDR protocol is taken into account for the simulation results shown in Figure 3.5.

Again the PDR protocol outperforms the slotted ring protocol for all loads and outperforms the DQDB for medium to high loads.

The slotted ring protocol always performs better when the message lengths are fixed and fit into the slots without causing fragmentation, as compared to variable length messages. Hence in the following simulation, a constant message length of 160 bits is used instead of an exponential distribution. Based on the standard DQDB protocol, a message that requires only one slot to transmit incurs a message overhead of only 192 bits instead of 224 bits. Therefore, the fixed message length of 160 bits was chosen so that the message can be transmitted using one slot with no fragmentation.

The simulation model for this case is based on a network of 100 nodes with a total ring delay of 2000 bits (Total ring delay for the PDR protocol is 2300 bits). The number of slots/tokens used is equal to 4. As shown in Figure 3.6, the slotted ring protocol outperforms the PDR protocol only when the throughput is less than 0.1. At higher loads, the PDR protocol still performs better with a maximum throughput that is almost twice that of the slotted ring.

In our next simulation, we compare the performance of the PDR protocol with DQDB, in a large network and with a fixed message length. The message length used is 1184 bits and requires exactly 4 slots to transmit the message. The ring consists of 50 nodes with a total ring delay of 5000 bits, including the 3-bit delay at the nodes for the PDR protocol. The number of slots/tokens

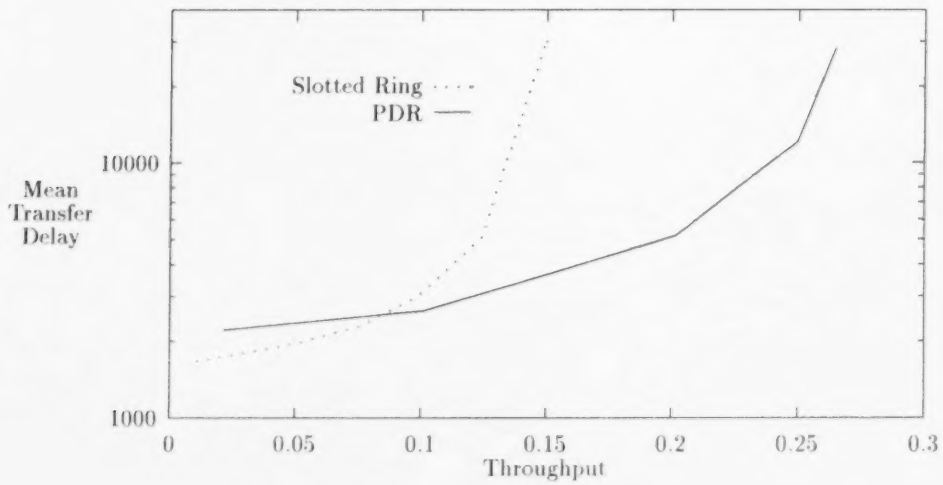


Figure 3.6: Comparison with Slotted ring: R=2000, m=160 (constant)

used is equal to 11. Figure 3.7 shows the result of this simulation. Under light and medium loads, DQDB outperforms the PDR protocol. But as the load is increased beyond 0.5, the PDR protocol still manages to perform better.

We have shown that even under conditions that are ideal for both the slotted ring and DQDB, the PDR protocol still achieves a better performance at the higher loads. Under conditions that are more realistic, that is, message lengths are exponentially distributed instead of fixed, the PDR protocol on the whole achieves a better performance, especially under heavy loads, for both short and long network spans.

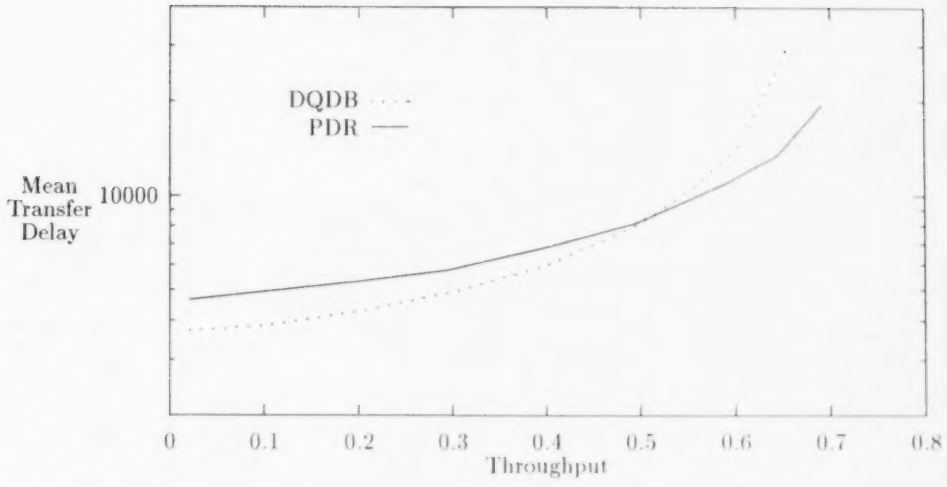


Figure 3.7: Comparison with DQDB: R=5000, m=1181 (constant)

3.5 Implementing fairness

In Chapter 2.4, fairness in the distribution of bandwidth utilization was discussed and a fairness mechanism was proposed for the PDR protocol. The graph in Figure 3.8 looks at the performance of the PDR protocol operating with 8 tokens with and without the fairness mechanism, and also using a few different values of *max*[1].

The value of *max* used in the fairness mechanism affects the degree of performance degradation in the protocol due to limiting bandwidth for hungry stations. It also affects the speed of enforcing fairness in the distribution of

[1]The maximum number of bits between RESET messages.

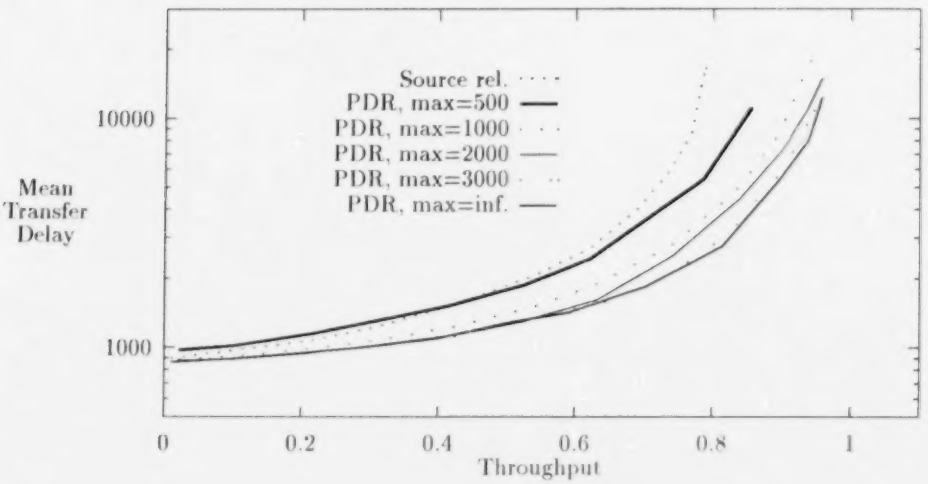


Figure 3.8: PDR protocol with and without fairness

bandwidth. A low value of *max* increases the frequency of sending the RESET control messages, therefore increases the amount of incurred overhead. As can be observed from Figure 3.8, using a *max* value of 500 bits results in a mean packet transfer time that is worse than that achievable when using the higher *max* values. However the reaction time of the fairness mechanism is faster which means that a starved node has to wait for a shorter period of time before it is granted the right to transmit. When a *max* value of 3000 is used, the performance of the protocol is only slightly affected at high loads. This shows that the reaction time of the mechanism is extremely slow and it is hardly activated when the load is low. Most importantly, the PDR protocol still outperforms the source release protocol with all four values of

	Utilization				max. transfer time (bits)
	average	std. dev.	max.	min.	
PDR, max=∞	0.021556	0.017401	0.023222	0.005214	547685
PDR, max=500	0.019943	0.000949	0.020281	0.019596	222330
PDR, max=1000	0.020440	0.000707	0.020651	0.020221	205631
PDR, max=2000	0.020784	0.000430	0.020908	0.020661	226564
PDR, max=3000	0.020936	0.000462	0.021051	0.020779	254017

Table 3.1: Bandwidth Utilization and Maximum Transfer Delay

max, except when $max = 500$ at light load. The choice of the value of *max* represents a tradeoff between performance degradation and reaction time. The fairness mechanism also guarantees an upper bound on the access delay and the value of *max* determines this upper bound.

To confirm that the mechanism ensures fairness in the distribution of bandwidth, the total amount of bandwidth utilized by each node is computed. The average and standard deviation values are calculated and used as our measure of fairness. The model simulates a highly loaded system (an approximate load level of 1.0) where the queues at all nodes are never empty, i.e., at least one packet is waiting to be transmitted at all times at every node. Subsequently there will be some nodes which will be starved, thus resulting in an unfair distribution of bandwidth. Table 3.1 and Figure 3.9 shows the statistics collected from the simulation.

Coupled with the average value, the standard deviation gives a good indication of the amount of variation in the distribution of bandwidth among the nodes. From Table 3.1, it clearly shows that the mechanism using any

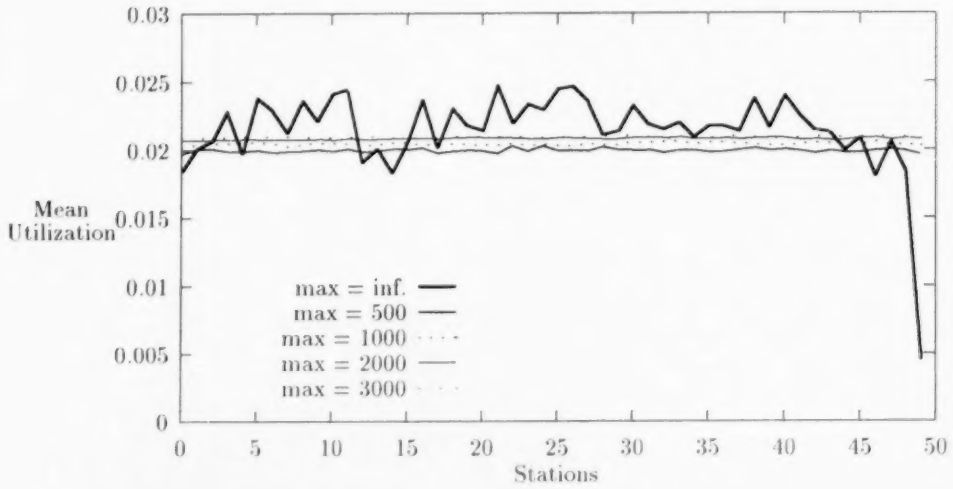


Figure 3.9: Distribution of bandwidth among the stations

of the four values of *max* has achieved fairness with a very low standard deviation compared to the PDR protocol without fairness. The maximum and minimum columns are used to show the actual bandwidth difference between the node that utilized the most bandwidth and the node that was most starved. The last column in the table is the maximum transfer time and is an indication of the upper bound achieved by implementing the fairness mechanism. Finally, Figure 3.9 shows the fluctuations in bandwidth distribution among the stations with and without the mechanism, and it clearly proves that fairness in the network has been achieved with the implementation of the fairness mechanism.

3.6 Upper bounds

At very light load, assuming that the tokens are equally spaced around the ring, the mean access delay is equal to $\frac{R}{2l}$ where R is the total ring delay and l is the number of tokens in the ring. In a worst case situation where the tokens are all clustered together acting almost as a single token, then the access delay will be approximately $\frac{R}{2}$.

At very heavy load, the source release protocol has an upper bound of $(N-1)m + R$ on the mean access delay where N is the number of tokens in the ring, m is the mean packet length and R is the total ring delay. For the PDR protocol, there is no such upper bound. In a worst case scenario, a node i captures all the tokens and transmits to the destination node j. At node j, the tokens are released and used immediately to transmit packets to node i, hence the two nodes i and j hog the ring completely and the queues at other nodes become unstable.

This problem can be solved by implementing the fairness mechanism which guarantees an upper bound on the access delay. If the fairness mechanism is implemented with maximum number of bits between RESET messages equal to max, then the access delay upper bound can be given by $(N-1)max + R$.

3.7 Summary

In this chapter, we have shown that the performance of the PDR protocol improves with increasing number of tokens but only up to a certain threshold. This is explained by the fact that as the number of tokens in the ring is increased, the reduction in access delay is overtaken by the amount of overhead incurred. This overhead becomes increasingly significant as the load is increased. The PDR protocol has been shown to outperform the FDDI, slotted ring, DQDB and the source-release multi-token ring protocol under various ring conditions, and for almost the entire throughput range. The fairness mechanism proposed in Chapter 2.4 has been proven to be effective in ensuring a fair distribution of bandwidth among the stations and setting an upper bound on the access delay.

Chapter 4

Priority Handling

4.1 Introduction

In a communication network, it is very common to have different types or classes of traffic which require different transmission priorities. For example, the different types of traffic can include data, voice, video and image. Data traffic can be further distinguished into batch and real-time. For token ring networks, protocols which implement transmission priorities can be classified into two main categories: 1) reservation based priority protocols and 2) timer based priority protocols. The following is a description of the different types of priority protocols which have been proposed in several papers.

4.2 IEEE 802.5 protocol

The IEEE 802.5 priority protocol [10] is a reservation based priority protocol and has been defined for low speed token ring networks. A node wishing to transmit at a certain priority level has to first capture an empty token. The node then transmits the packet and awaits the return of the token header before releasing the token. The token header has a field that indicates the priority level of the token and a node can only use an empty token if the priority level of its packet is higher than or equal to that of the token. The token format has another field which can be used by a node to make a priority reservation on a passing token. The procedure shown in Figure 4.1 is used by a source node to increment the priority level of a token. The node that increased the priority level of the token from i to j is responsible for reducing the priority back to i when the token returns at priority level j. A stack is used to store the previous priority level i so that a node may perform multiple increases in the priority level and the correct previous priority level can then be retrieved from the stack.

4.3 Timer-based Priority protocols

Releasing the token at the end of transmission is important in high speed token networks because the packet lengths are significantly shorter than the

```
If (token reservation j > token priority i) then begin
      push (i, j) onto stack
      set token priority = j
      set token reservation = 0
      release token
Else
      release token at the same priority and reservation
```

Figure 4.1: Algorithm for Priority Handling

ring propagation delays. For example, assume a token ring LAN where the total ring delay is equal to R and the mean packet length is equal to m, where $m \ll R$. If a token is released immediately after a packet transmission, then the service time is only equal to m. However if the token is released after one round trip delay, then the service time will be equal to R. It is clearly a much better utilization of bandwidth to release the token immediately.

The FDDI priority protocol [23] is a timer based priority protocol and has been defined for high speed token ring networks. In the FDDI protocol, an empty token is released immediately after the end of transmission instead of waiting for the header to return. Not having to wait for the header also means that the node releases the token before receiving any priority reservation, thus a reservation based priority system is not used. To implement priority in the FDDI protocol, each node has a timer that keeps track of the time between successive token arrivals (TRT - Token Rotation Timer). If the TRT has a

low value, it means that the token is rotating fast and thus indicating that the load is low. If the TRT has a high value then it indicates that the load is high.

Based on the timer values, a scale is constructed which indicates the different levels of load in the ring and subsequently also indicates the levels of priority. For example, let $t_0, t_1, \ldots, t_i, \ldots, t_{n-1}$ be timings on the scale such that $t_0 < t_1 < \cdots < t_i < \cdots < t_{n-1}$ where i is the priority level. A node with traffic of priority level i may transmit only if the TRT has a value $t < t_i$. Therefore traffic with a priority level of n, can be transmitted at all times. This timing scale is determined by some network management protocol based on bandwidth and delay requirements.

4.4 A Modified Reservation-based Priority protocol

The priority protocol described in [6] is a reservation based priority protocol that has been defined for a high speed token ring network. A token release procedure similar to that used in the FDDI protocol is adopted, i.e. the token is released at the end of packet transmission. A node can make a priority reservation in three ways:

1. Wait for a packet header to pass by and make the priority reservation (request) in the header as in the IEEE 802.5 protocol.

2. Create and transmit a short priority reservation request whenever the ring is sensed idle.

3. If the node is in the process of transmitting a packet when a higher priority reservation request (with preemptive capability) arrives, preempt the current transmission and transmit the priority reservation request.

When a node wants to transmit a packet of priority i, it has to capture an empty token of priority $j \leq i$. If during the transmission, the node receives several priority reservation requests (assuming none has preemptive capability), upon completing the transmission it will release the token at the highest priority k chosen among the requests. Again the same node is responsible for setting the priority back to the previous value j when the empty token returns at priority k. The same stacking/unstacking technique as in the IEEE 802.5 priority protocol is used. If no priority requests were received, then the token will be released at the same priority level j. If a node receives a priority request when it is not holding the token, it will just pass the request on to the next node.

4.5 A Packet Preemption protocol

The priority protocols discussed so far are all non-preemptable, that is, if a high priority packet arrives after a lower priority packet has already captured the token, then the higher priority packet has to wait for the lower priority packet to complete its transmission. On the other hand, a preemptable priority protocol will terminate the transmission of the lower priority packet to allow the higher priority packet to be transmitted first.

There is currently no preemptable priority protocol existing in token ring networks, so in [26], a simulated preemptable reservation based priority protocol was compared with a non-preemptable protocol (IEEE 802.5). The paper shows that a preemptable protocol provides better service to high priority packets in the sense that the delay for high priority packets is independent of the load. A non-preemptable protocol however takes into account fairness to low priority traffic and therefore service to high priority traffic degrades as the load increases.

4.6 Priority handling in PDR protocol

In a multiple token ring network, there is a difference between updating the priority of a token and updating the priority of the ring. Updating the ring priority means that all tokens possess the same priority and any updating

should affect all tokens at the same time, whereas updating just the token priority means different tokens may have different priority levels updated at different times.

In the PDR protocol, it is not possible to update the priority of all the tokens at the same time. The time it takes for the ring priority to be incremented is approximately equal to one round trip delay. If the high priority packet has a very short packet length, then it is much more efficient to update only the token priority because it would most probably require only one or two tokens to transmit the packet. If the high priority packet is a long packet, then more tokens will have their priorities raised and eventually will be the same as having the ring priority raised. Therefore the priority protocol to be defined will update the priority of the tokens, not the ring.

A reservation-based priority scheme is prefered over the timer-based priority scheme in the PDR protocol because the presence of multiple tokens in the ring will present certain problems with a timer-based priority scheme:

- Different TRTs must be used for the different tokens to keep track of the token rotation times.
- The TRTs for the different tokens may have different values. A node that has started a packet transmission when the TRT value of the first token is low, may not be able to continue its transmission if the TRT value of the next available token is high.

- The TRT for the same token may have different values at different nodes, hence, the problem of identifying the addresses of a continuation transmission may be hard to resolve.

The operation of the PDR priority protocol to be defined is reservation-based and is similar to the priority protocol defined in IEEE 802.5 [10]. Referring to the token format in Figure 2.1, the **Priority** field stores the priority level of the token and the **Reservation** field is used by the nodes to make priority reservations on the token. A node wishing to transmit a packet of priority i has to capture an empty token of priority $j \leq i$. Priority reservations can be made in the headers of all **Normal** tokens provided that another higher priority packet has not already filled the **Reservation** field. Reservations are not made in **Cut** tokens because they might be removed from the ring before they reach a node that is releasing a token. The number of priority reservations made by the node is dependent on the number of tokens that is required to transmit the packet.

When the source node is responsible for releasing the token, it will not receive any priority reservations before the end of its transmission, hence the token will be released at the same priority level (Note that tokens are set to **Cut** if they are source released). If the length of the packets are very short such that only one token is required to transmit each packet, then the probability of tokens being released by their source nodes is very high. Stations wishing to send priority reservations will have to wait for a longer

period of time before a **Normal** token arrives. For this priority scheme to be effective, the number of tokens used should be sufficiently large such that the average length of the packets is longer than the average bandwidth associated with the tokens. Subsequently, the probability of tokens being released by their destination nodes will also be high and priority reservations can be sent within a shorter period of time.

As explained in Chapter 2.2.3, a node waiting to resume its packet transmission, is expected to use the next empty token that arrives. However, if the priority of the next empty token is higher than the packet's priority, then the token cannot be used. In order for the Table Management Process to correctly identify the addresses of a continuation transmission, the following modifications to the protocol are required:

- The priority of the packet that is transmitted must be stored in the **Priority** field of the *Node Table* (Figure 2.3).

- The Table Management Process (Figure 2.7) must be modified to search for the next **Active** node in the *Node Table* with the packet priority greater than or equal to the priority of the current token, instead of just searching for the next **Active** node.

Since there are multiple tokens in the ring, a node may increase the priority of several tokens and several nodes may increase the priority of the same token. In the *Token Table*, the field **Priority** contains a pointer to

a different stack for each token. The use of the stacks is similar to that in the IEEE 802.5 priority protocol and so is the procedure for increasing and decreasing the token priorities.

4.7 Performance Measure

A simulated model of the priority protocol has been used to examine the effectiveness and performance of the priority protocol. The token length used in the simulation is increased from 10 to 12 bits to accomodate the extra priority fields while the packet overhead remains at 12 bits. The total ring delay is 1000 bits with 50 equally spaced nodes connected to the ring. The packet lengths are exponentially distributed with a mean of 200 bits and packet interarrival times are also exponentially distributed. Using only two priority levels of LOW and HIGH, simulation results are obtained for a network operating with 2 and 8 tokens. The offered traffic load consists of approximately 50 percent LOW priority packets and 50 percent HIGH priority packets. Figure 4.2 shows the different service levels provided by the protocol to packets at the two priority levels. For high priority packets with either 2 or 8 tokens, the mean transfer delay increases slowly with increasing load. On the other hand, service to the low priority packets starts to degrade rapidly even at medium loads.

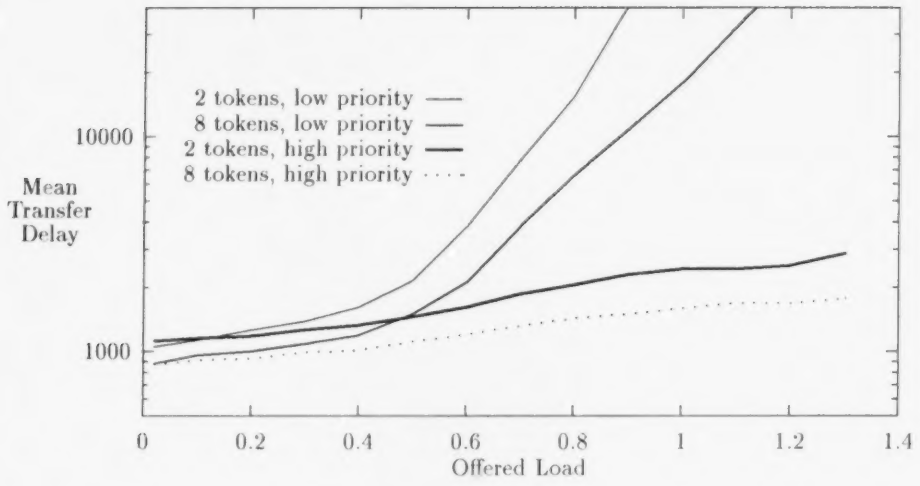


Figure 4.2: Priority scheme with 2 priority levels using 2 and 8 tokens

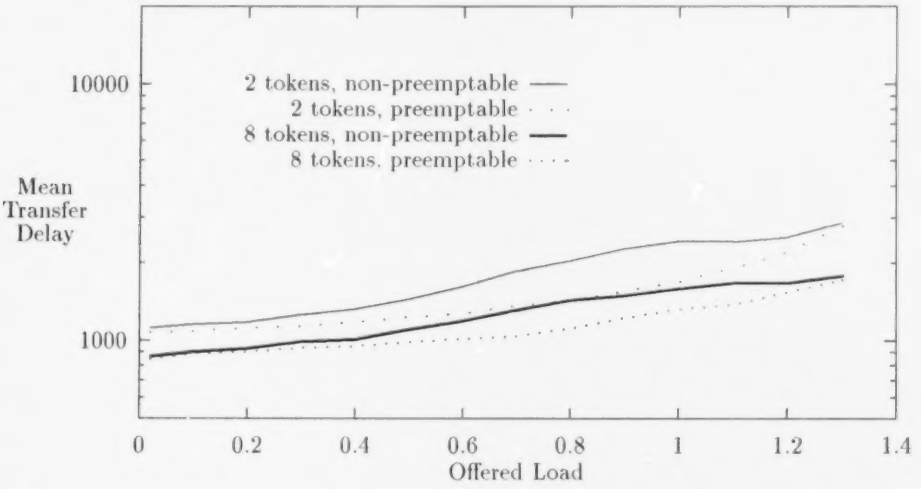


Figure 4.3: Preemtable and non-preemtable priority schemes with 2 and 8 tokens

The priority scheme that we have proposed is non-preemptable[1]. To evaluate the effectiveness of our priority protocol, we have to compare its performance to that of a preemptable protocol. The preemptable protocol is simulated by completely eliminating the transmission of LOW priority packets. The composition of offered load remains the same at 50 percent LOW priority traffic and 50 percent HIGH priority traffic. The simulation result is shown in Figure 4.3. As the load is increased from low to medium, the performance of the PDR protocol starts to degrade, relative to the preemptable protocol. However as the load continues to increase, the performance of both protocols becomes comparable. This is because at high loads, HIGH priority traffic in the PDR protocol has managed to shut out most of the LOW priority transmissions. We also observe that the PDR protocol operating with 8 tokens provides better service to its high priority traffic than with 2 tokens. This is to be expected, as explained earlier, a larger number of tokens will provide a faster response to priority reservation requests.

4.8 Summary

In this chapter, we have reviewed several different priority schemes and a reservation-based multi-level priority scheme has been proposed for the PDR

[1]Refer to Chapter 4.5 for definition of preemptable and non-preemptable priority schemes.

protocol. The proposed scheme can be easily implemented by adding the two fields **Priority** and **Reservation** in the token header (Figure 2.1), and the column **Priority** in both the Token Table (Figure 2.4) and Node Table (Figure 2.3). The procedure involved in the implementation of the priority scheme is simple and requires minimal added computation at the nodes. The results obtained from the simulation model have shown that the PDR priority protocol is able to achieve its purpose of providing better service to the high priority packets. The priority scheme is most effective at high loads and with a large number of tokens.

Chapter 5

Dynamic Changing of the Number of Tokens

5.1 Introduction

In this section, we further examine the PDR protocol by exploring the effects of changing some of the parameters on the protocol performance. The parameter that we are most interested in is the number of tokens operating in the ring that achieves the best performance, this number will be called the optimal token number. Again, the sole performance measure used will be the mean transfer delay.

It is evident from the study in Chapter 3 that as the operating conditions are changed, the optimal token number is also changed, this motivates

equipping the protocol with the capability of changing the number of tokens in the ring dynamically. To this end, an algorithm that can compute the optimal token number must be provided. However devising such an algorithm is difficult as it involves finding the token number changing points based on continuously changing ring conditions. Identifying such points can be based on a learning process and is beyond the scope of this thesis.

We shall derive an equation to find the mean transfer delay under very light load based on the following assumptions:

1. The tokens are equally spaced around the ring. Hence, the mean access delay is $\frac{R}{2l}$ and the average bandwidth associated with each token, b is equal to $\frac{R-tl}{l}$ where R is the total ring delay, l is the number of tokens and t is the token length.

2. If a packet requires more than one token for transmission, then only the first token has a mean access delay of $\frac{R}{2l}$. All subsequent tokens have a mean access delay of zero.

3. The queueing time[1] for each packet is equal to zero.

4. Since the destination of a packet is uniformly distributed, the mean propagation delay is equal to $\frac{R}{2}$.

[1]The time from the arrival of the packet until the packet is at the head of the queue.

Let the overhead associated with each packet be denoted by h and the mean packet length be denoted by $\bar{m}$. The mean transfer delay under very light load can now be represented by the following equation:

$$\bar{y} = \frac{R}{2l} + \bar{m} + h + \frac{R}{2} + [\frac{\bar{m}+h}{b} + 1]t = \frac{R}{2l} + \bar{m} + h + \frac{R}{2} + [\frac{(\bar{m}+h)l}{R-tl} + 1]t \quad (5.1)$$

where $[\frac{\bar{m}+h}{b} + 1]$ is the approximate average number of tokens required for the transmission of a packet. This is based on the assumption that the tokens are equally spaced around the ring.

Using the above equation, the mean transfer delay is computed for increasing integer values of l starting from $l = 1$. As the value of l increases, the value of y decreases until it reaches a minimum value (threshold) and then starts to increase. This phenomena can be explained by the fact that the first term in y decreases with l, while the last term increases with l. At the threshold, the corresponding value of l will be the optimal token number under very light load. This computed token number is also the upper bound on the optimal number of tokens that should be operating in the ring for the different loads. This is because as the load increases, the access delay for subsequent tokens and the queueing time for each packet will no longer be equal to zero. Therefore the optimal number of tokens will decrease with increasing load.

The find the minimum value (threshold) of $\bar{y}$, we differentiate Equation

5.1, to obtain the following expression:

$$\frac{d\bar{y}}{dl} = \frac{-R}{2l^2} + \frac{Rt(\bar{m}+h)}{(R-tl)^2}$$

To obtain the minimum value for $\bar{y}$, let $\frac{d\bar{y}}{dl} = 0$

$$\Rightarrow \frac{-R}{2l^2} + \frac{Rt(\bar{m}+h)}{(R-tl)^2} = 0$$

The upper bound for the optimal number of tokens under light load can then be computed from the following equation:

$$l = \frac{Ra}{1+ta} \tag{5.2}$$

where $a = \sqrt{\frac{1}{2t(\bar{m}+h)}}$.

Having computed the upper bound, we will use the simulator to obtain the corresponding optimal token numbers for the different loads. The following are four examples with slightly different parameters used to show the results of our approach to finding the optimal token numbers.

Case 1

The parameters used are:

Number of nodes,	N = 50
Total ring delay,	R = 1000 bits
Mean packet length,	m = 200 bits (exponential distribution)
Packet overhead,	h = 12 bits
Token length,	t = 10 bits

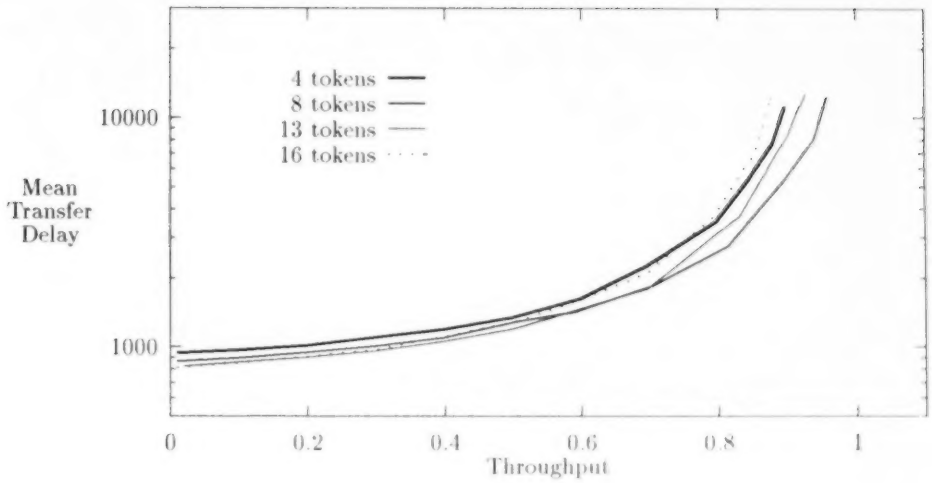


Figure 5.1: PDR protocol with 4, 8, 13 and 16 tokens: R=1000, m=200 (exp)

From Equation 5.2, the optimal token number is estimated to be equal to 13. This number is the approximate upper bound for the optimal number of tokens. Figure 5.1 shows simulation results using 4, 8, 13 and 16 tokens. The graph shows that using 13 tokens in the ring results in the lowest mean transfer delay from the throughput value of 0.0 until about 0.7. As the load increases beyond 0.7, the network with 8 tokens begins to outperform the rest and generates a maximum throughput of almost 1.0. Note that when 16 tokens are used, the performance of the protocol does not show any improvement. Instead, it shows rapid degradation with increasing load. This corresponds to our approximate upper bound value of 13 for optimal token numbers.

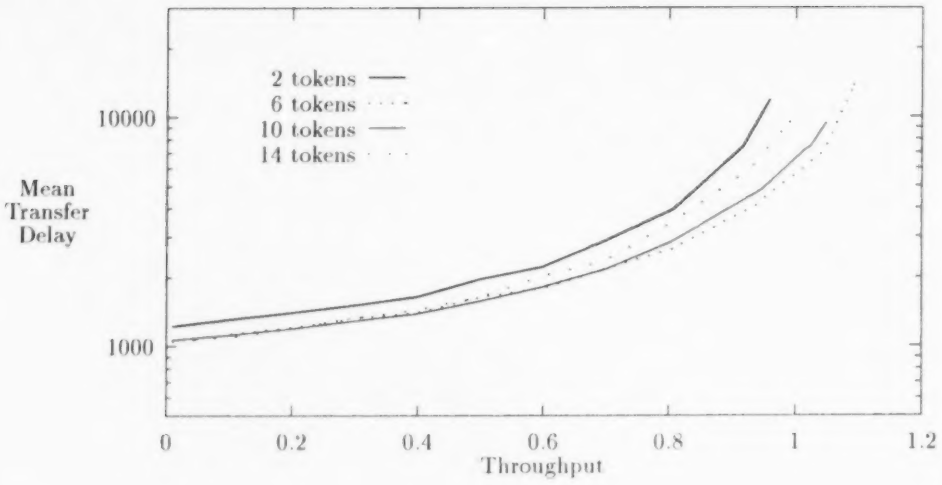


Figure 5.2: PDR protocol with 2, 6, 10 and 14 tokens: R=1000, m=400 (exp)

Case 2

In the second case, the mean packet length m, is increased from 200 bits to 400 bits with an exponential distribution while the rest of the parameters remain unchanged. Similar to the first case, the upper bound on the token numbers can be obtained from Equation 5.2. The optimal token number at light load in this case is approximated to be equal to 10. Figure 5.2 shows the results of our simulation using 2, 6, 10 and 14 tokens. From the figure, the optimal token number at light load is equal to 10 which corresponds to the computed upper bound, and remains at 10 for throughput values from 0.0 to 0.4. For throughput values greater than 0.4, the optimal token number becomes 6.

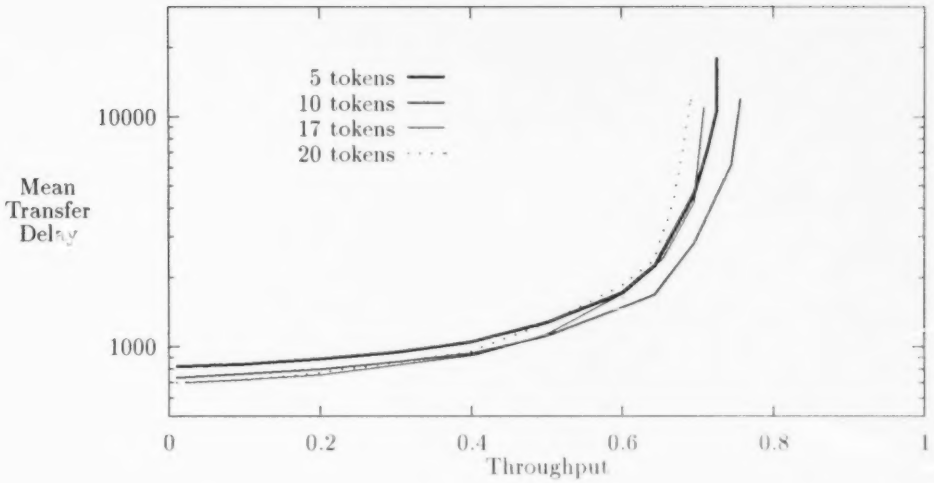


Figure 5.3: PDR protocol with 5, 10, 17 and 20 tokens: R=1000, m=100 (exp)

Case 3

In case 3, the mean packet length m, is decreased to 100 bits with the other parameters remaining constant. The upper bound on the token numbers obtained from Equation 5.2 is estimated to be 17. Figure 5.3 shows the results of our simulation using 5, 10, 17 and 20 tokens. Similar to the previous cases, the optimal token number is 17 under light load and decreases to 10 when the load exceeds 0.4. Again the computed upper bound of 17 tokens is shown to be correct.

Case 4

In this last example, the number of nodes N, is increased from 50 to 100.

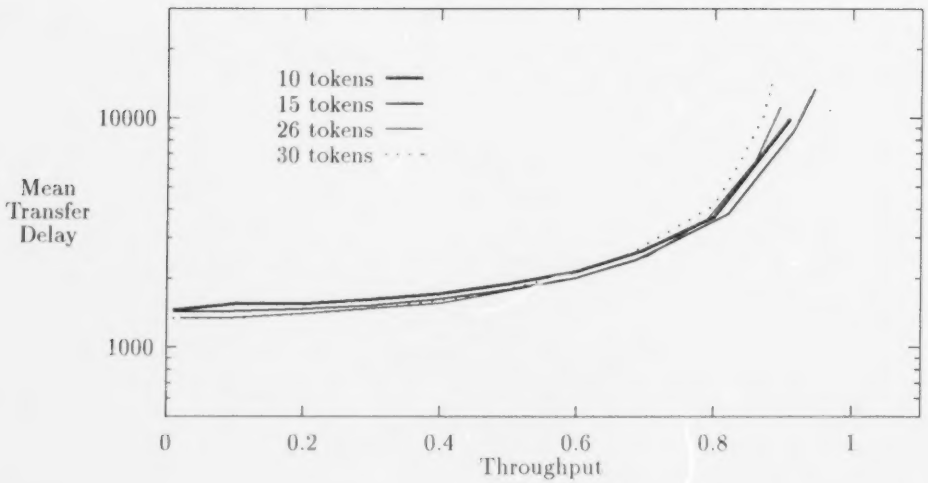


Figure 5.4: PDR protocol with 10, 15, 26 and 30 tokens: R=2000, m=200 (exp)

The distance between nodes remains unchanged hence the total ring delay R becomes 2000 bits. The rest of the parameters are similar to those for Case 1. The upper bound on the token numbers obtained from Equation 5.2 is approximately equal to 26. Figure 5.4 shows the results of our simulation using 10, 15, 26 and 30 tokens. Again we observe an optimal token number of 26 at light load and a different optimal token number of 15 at heavy load when throughput exceeds 0.4.

In the above examples, the graphs only show the performance results of a few selected token number values, hence the optimal token numbers obtained are just approximations. The intension is to show that the optimal

token number does change with changing parameters. The next section capitalizes on this feature by equipping the PDR protocol with the capability of dynamically changing the number of tokens.

5.2 Protocol Description

In the following, we assume that there exists a monitor station that, based on the network parameters, will compute the optimal number of tokens that should be operating in the ring. Only a monitor station can initiate the removal and addition of tokens. The scheme can be implemented with more than one station acting as monitor stations. However, if the monitor stations are not coordinated, oscillations in the number of tokens may develop. In addition, optimal solutions may not be reached due to different monitor stations having contradictory decisions.

The number of tokens can be changed dynamically by making use of the existing priority handling procedures in the PDR protocol. We also assume that the priority scheme in the protocol uses a 3-bit code to indicate the different priority levels. The two highest levels of priority code will then be reserved to represent the two control messages ADD and REMOVE. With a 3-bit code, the total number of priority levels that the packets can have will now be reduced from eight to six levels. The two highest levels of priority were chosen to represent the control messages in order to prevent stations

from using the token while it is holding a control message. We now proceed to describe the token removal and token addition procedures.

Token Removal

The monitor station can initiate the removal of a token by setting the **priority reservation** of the token to the control message REMOVE. The first node to use and release this token will update the **priority** field of the token to REMOVE and update the priority stack in the same manner as incrementing a normal priority level[2].

As the token propagates around the ring, all nodes detecting the REMOVE control message will set the token entry in the *Token Table* to **Inactive**. When the token returns to the node which updated the token **priority**, the token entry in the *Token Table* will be set to **Inactive** and the token will be removed from the ring.

Once the entry in the *Token Table* has been set to **Inactive**, the Token Management Process will ignore the entry and the protocol execution can proceed normally.

Token Addition

Three conditions must be satisfied before a token can be added to the ring:

1. the token must have been removed previously,

[2]Refer to Chapter 4 for a detailed description of priority handling.

2. the token before it must be present in the ring and

3. there must be sufficient bandwidth for the token to be added.

There is a maximum number of tokens that are allowed to operate in the ring and the number of entries in the *Token Table* must be equal to this maximum number. Hence the monitor station can add a token if and only if

1. there exists an **Inactive** entry in the *Token Table*,

2. there exists an **Active** entry before the **Inactive** one and

3. the associated bandwidth of the **Active** entry is greater than one token length.

The process can be initiated by the monitor station by setting the **priority reservation** field of the **Active** token to the control message ADD. The first node to release this token will update the *Token Table* and the token **priority** to ADD using the same procedure as updating a priority level. Then two empty tokens are released with the first token holding the control message.

All nodes detecting the ADD control message will set the next entry in the *Token Table* to **Active** and will also update the fields of this new **Active** entry. After propagating a complete round back to the first node, the ADD message is removed by setting the token **priority** back to zero.

```
Step 1.  If ( token Reservation = REMOVE ) then
             push (0,REMOVE) onto stack
             set token Priority = REMOVE
             set token Reservation = 0
             release token
         end

Step 2.  If ( token Reservation = ADD ) then
             push (i,ADD) onto stack ( where i=token priority )
             set token Priority = ADD
             set token Reservation = 0
             release token
             append empty token with Priority = i
         end
```

Figure 5.5: Algorithm for addition/removal of tokens in Receiver process

This scheme of removing and adding tokens can be implemented by simply adding the procedures shown in Figures 5.5 and 5.6 to the Receiver and Table Management processes respectively. The Transmitter process remains unchanged.

The reaction time, or the time taken for a token to be added or removed since the control message was first initiated by the monitor station depends on two factors, the total ring delay and the ring utilization.

Note that the monitor station can send a control message in the **Reservation** field of two types of tokens: an **Empty** token and a used **Normal** token. If it is sent in a used token, then the token will be released with the control message in its **Priority** field in less than one round trip delay. After

```
Step 1.   If ( token Priority = REMOVE ) then
              set entry in Token Table to Inactive
              retrieve values (i, j) from stack
              if ( j = REMOVE ) then
                   remove token from ring
              end

Step 2.   If ( token Priority = ADD ) then
              set next entry in Token Table to Active
              update fields of next entry
              retrieve values (i, j) from stack
              if ( j = ADD ) then
                   set token Priority = 0
              end
```

Figure 5.6: Algorithm for addition/removal of tokens in Table Management process

its release, it will take another round trip delay to inform all nodes of the change. Hence the reaction time is less than $2R$, where R is the total ring delay.

However if the control message is sent in an **Empty** token, then there is no upper bound on the reaction time. Under heavy load, the probability of an **Empty** token being captured by a busy node is higher, hence the reaction time will be shorter, while under light load the reaction time will be longer. To set an upper bound on the reaction time, the monitor station will send the control message in the **Reservation** field of the **Empty** token and set the token status to **SED** with both the source and destination addresses equal to the monitor station. When the token returns, the monitor station

will release the token with the control message in its **Priority** field. The reaction time in this case is exactly equal to $2R$.

The overhead incurred everytime a token is added or removed is equal to the bandwidth associated with the token holding the control message. This is because whenever a token is holding a control message, it cannot be used by any of the nodes and the control message is held for a length of time equal to R.

5.3 Simulation Results

A simulator was written to examine the effectiveness of the dynamic token addition/removal procedure implemented in the PDR protocol. Only the throughput will be changed during each simulation run, while the following parameters remain constant.

Number of nodes	= 50
Total ring delay	= 1000 bits
Packet overhead	= 12 bits
Token length	= 10 bits
Mean packet length	= 100 bits (exponential distribution)

For throughput values of less than 0.3, the number of tokens used will be 20, for throughput values between 0.3 and 0.6, 15 tokens will be used and for throughput values greater than 0.6, 10 tokens will be used. Station number

0 is assigned as the monitor station and it is responsible for the addition and removal of tokens. It is assumed that the monitor station is capable of estimating the throughput by monitoring ring activity over a certain period of time. This period of time will be called the **window**.

It is obvious that the size of this window affects the responsiveness of the token addition and removal procedures. Choosing a suitable window size depends on factors like:

- the reaction time [3],
- the type of traffic generated, e.g. bursty or continuous,
- the amount of overlap between consecutive windows and
- the frequency of token adding and removing.

If the window size is too small, fluctuations in throughput triggers the unneccessary addition and removal of tokens which results in a degradation of performance. If it is too large, then the change in the number of tokens might be too slow to have any positive effect on the performance.

The performance of the protocol will be examined using a window size of $50R$ together with the following settings:

- The packet interarrival time is exponentially distributed.

[3]Time taken from the sending of the control message to the actual addition or removal of token.

- The throughput is changed every $1000R$ where R is the total ring delay, this will be called a cycle. If the ring is operating at a speed of 100 megabits per second, then the length of the cycle is about 0.01 second.
- The throughput for each cycle will be randomly generated at the beginning of each cycle.
- At the end of the cycle, the mean throughput and mean transfer delay will be computed.
- There will be an overlap in successive windows equal to half a window size.

From Figure 5.7, the PDR protocol implemented with dynamic changing of number of tokens (PDR-DC protocol) comes very close to the performance of the protocols with fixed token numbers of 10, 15 and 20. The ideal case is to have the PDR-DC protocol achieve a performance level that is the best among the protocols with fixed token numbers over the entire range of throughput values.

Next, we increase the window size from $50R$ to $200R$ to observe the effect of the window size on the performance of the PDR-DC protocol. From the results shown in Figure 5.8, the performance of the network seems to degrade slightly. This phenomenon may be explained by the fact that the window size used in this case is too large. With a large window, the reaction to changing

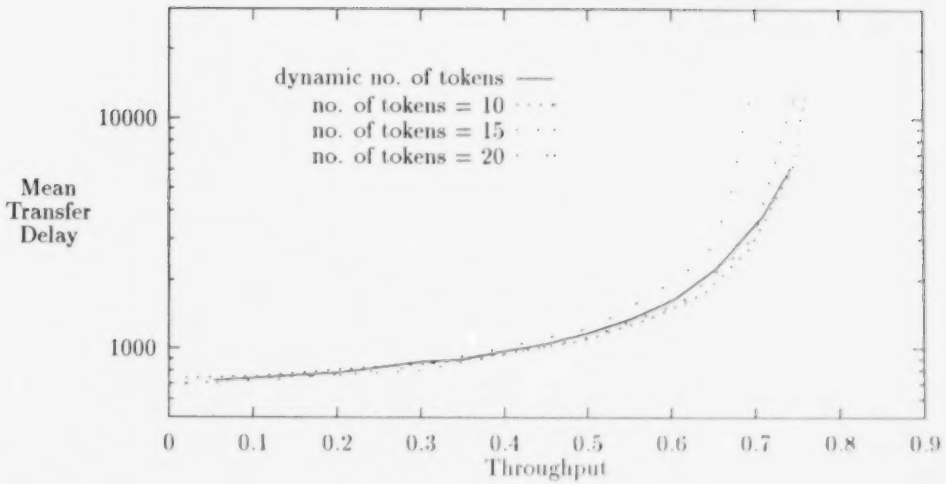


Figure 5.7: PDR-DC protocol with window size = $50R$

ring conditions is slow and therefore has resulted in a worse performance as compared to the previous case.

Based on the results we have seen so far, the overhead incurred in the addition and removal of tokens has caused only a slight overall performance degradation. The PDR-DC protocol with its capability to maintain an optimal number of tokens in the ring at all times, has been shown to achieve an overall performance that is better than using a fixed number of tokens. In a practical implementation of the PDR-DC protocol, a procedure is required to give an accurate estimate of the current ring parameters and the choice of window size depends on several factors which was discussed earlier.

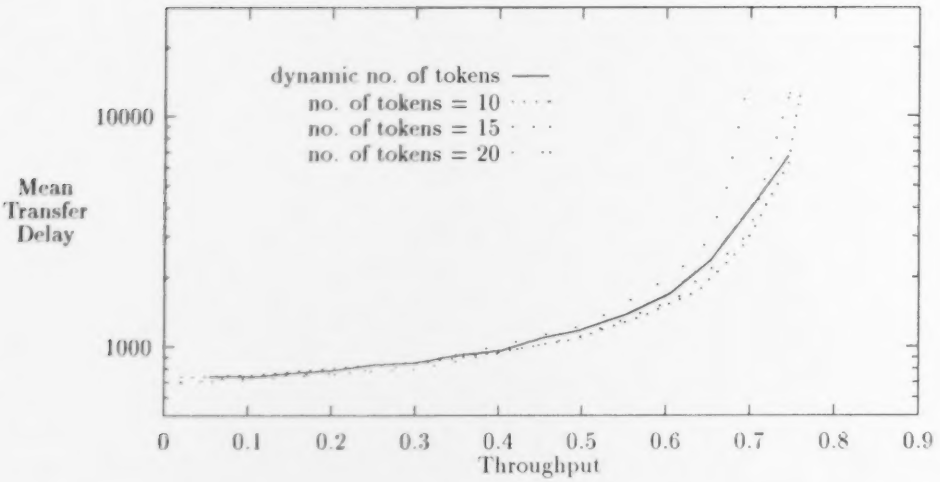


Figure 5.8: PDR-DC protocol with window size = $200R$

5.4 Summary

In this chapter, we have derived an equation (Equation 5.2) that can be used to determine the upper bound for the optimal number of tokens, given the network parameters. Simulation results were then used to check the correctness of this computed upper bound and also estimate the optimal number of tokens for the different load levels. The simulation results also motivated the introduction of a new feature that enables the PDR protocol to dynamically change the number of tokens operating in the ring based on the current network conditions. This feature can be easily implemented on top of the priority scheme presented in Chapter 4. Again the simulation model

was used to check the effectiveness of this new feature. The performance of the protocol with its dynamic token addition/removal capabilities was shown to come very close to the ideal case when the correct window size is used.

Chapter 6

Conclusions

By combining the features of the multiple-token ring protocol by [11] with the destination release technique by [28], a new protocol has been defined that allows a token to be released by the source node if there is enough room to release the token, or by the destination node otherwise. The aim of this approach is to utilize the channel capacity to its fullest by releasing a used token as soon as possible. A detailed description of the new Partial-Destination-Release Multi-token (PDR) ring protocol was presented in Chapter 2 which includes a description of the fairness mechanism that has been proposed to overcome the problem of unfair bandwidth distribution among the stations. Chapters 4 and 5 then proceed to describe how the PDR protocol can be equipped with the capability to handle multiple levels of priority transmission and the unique capability to dynamically change the number of tokens

in the ring.

It has been shown from the simulation results in Chapter 3 that the PDR protocol with its partial destination release feature outperforms the source release multiple-token protocol proposed in [11] as well as other popular network protocols like FDDI, the conventional slotted-ring and DQDB. The improvement in performance is seen over the low, medium and high traffic levels. The presence of multiple tokens in the ring results in a more significant performance improvement as the proportion of the packet length to the total ring delay becomes smaller. This is particularly important in high speed networks as this proportion decreases with increasing speeds in networks.

However the complexity of the protocol has compromised the overall reliability of the network as the occurrence of errors in the token headers or the tables may prove to be irrecoverable. The protocol requires a lot more storage space and computations performed at each station than the normal single token ring but is still comparable to that of the source release multiple-token protocol. The amount of computation required is also dependent on the size of the network, as networks with large numbers of stations will have larger tables which take more time to process.

The issue of fairness in a destination release protocol has been addressed in Chapter 2 and a fairness mechanism similar to that implemented in the Orwell protocol can be easily implemented in the PDR protocol. Another important and common requirement in networks today is the ability to ac-

comodate different types of traffic by having different transmission priority levels. By adding two more fields in the token header and a stack associated with each token, the PDR protocol is thus able to provide multiple levels of priority transmission.

Based on the implementation of priority handling in the protocol, an interesting feature can be easily added, viz., the dynamic changing of the number of tokens in the ring. The amount of extra computation required to add and remove tokens is insignificant, but the algorithm used to determine the optimal number of tokens that should be in the ring is difficult to derive. Instead, we used simulation results obtained from Chapter 3 for a few specific cases and based on those results, the adaptive token number scheme was implemented in our simulation model. The ideal case would be to achieve the lowest mean transfer delay over the entire throughput range compared to protocols with fixed number of tokens. The result we obtained from the simulation is very close to the ideal case and the extra computation required to implement this feature is insignificant. Further investigations can be done for several more cases, using different approaches to computing the current ring parameters and perhaps an approximate solution to computing the optimal number of tokens.

Several modifications or extensions to the PDR protocol are possible, for example:

- The inclusion of source or destination address in the token header.

This would reduce the complexity of the protocol and at the same time increase the reliability of the network. Of course the overhead associated with each token will be increased.

- Modifying the protocol to allow the last token used in a packet transmission to propagate once around the ring before it is released by the source node.

  This would enable the destinations to piggyback acknowledgements onto the last tokens back to the source node. The modification required to accomodate this change is relatively simple and only involves setting the appropriate status and type bits of the last token to indicate an **End** of packet transmission and a **Source** released **Normal** token.

- Adapting the protocol to a dual counter-rotating token ring network. Having two rings instead of one will definitely improve the reliability of the network especially in the case of single node failures. Throughput may be increased twofold, access delays and mean propagation delays can be reduced. However, the computation of destination addresses for continuation transmissions may prove to be significantly more complex since packets can now be transmitted in two directions.

For a practical implementation of the PDR protocol, a modification of the protocol to include the source address would seem more feasible as it reduces the complexity and improves the reliability. Modifying the protocol

to allow the acknowledgements of packets is also recommended especially when a large number of tokens is used.

Bibliography

[1] J.L. Adams and R.M. Falconer. 'ORWELL': A protocol for carrying integrated services on a digital communications ring. *Electronics Letters*, 20(23):299–262, October 1984.

[2] B.W. Arden and H. Lee. Analysis of chordal ring. In *IEEE Transactions on Computers*, volume C-30, pages 291–295, April 1968.

[3] B.V. Arem and E.A.V. Doorn. Analysis of a queueing model for slotted ring networks. In *ITC Specialist Seminar*, Adelaide, 1989. paper no. 8.5.

[4] A. Bondavalli and L. Strigini. DSDR: A fair and efficient access protocol for ring-topology MANS. In *Proceedings of the 1991 IEEE INFOCOM*, 1991.

[5] I. Cidon and Y. Ofek. Distributed fairness algorithms for local area networks with concurrent transmissions. In *3rd International Workshop on Distributed Algorithms*, 1989.

[6] A. Goyal and D. Dias. Performance of priority protocols on high speed token ring networks. In *Proc. of Int. Conf. on Data Comm. Systems and their Perf.*, pages 25–34, Rio de Janiero, Brazil, June 1987.

[7] E.R. Hafner, Z.Nendal, and M.Tschanz. A digital loop communication system. *IEEE Transactions on Communication*, C-22(6):877–881, June 1974.

[8] A. Hopper and R.M. Needham. The Cambridge Fast Ring networking system. *IEEE Transactions on Computers*, 37(10):1214–1223, October 1988.

[9] A. Hopper and R.C. Williamson. Design and use of an integrated Cambridge Ring. *IEEE Journal on Selected Areas in Communication*, SAC-1(5):775–784, November 1983.

[10] *ANSI/IEEE Standard 802.5, Token Ring Access Method and Physical Layer Specifications.*

[11] A. E. Kamal. On the use of multiple tokens on ring networks. In *Proceedings of the IEEE INFOCOM*, pages 15–22, 1990.

[12] A. E. Kamal. An algorithm for the efficient utilization of bandwidth in the slotted ring. *IEEE Transactions on Computers*, 1992. To appear.

[13] A.E. Kamal and V.C. Hamacher. Approximate analysis of non-exhaustive multiserver polling systems with applications to local area networks. *Computer Networks and ISDN Systems*, 17, 1989.

[14] A.E. Kamal and V.C. Hamacher. Utilizing bandwidth sharing in the slotted ring. *IEEE Transactions on Computers*, 39(3):289–299, March 1990.

[15] F.R. Kschischang and M.L. Molle. The helical window token ring. *IEEE Transactions on Information Theory*, 35(3):626–636, May 1989.

[16] M.A. Marsan, L.F. Moraes, S. Donatelli, and F. Neri. Analysis of symmetric nonexhaustive polling with multiple servers. In *IEEE INFOCOM*, volume 1, San Francisco, June 1990.

[17] J.J. Metzner. A high efficiency acknowledgement protocol for the slotted Pierce Ring. In *Proceedings IEEE INFOCOM*, pages 333–339, 1985.

[18] R.J.T. Morris and Y.T. Wang. Some results for multi-queue systems with multiple cyclic servers. In *Performance of Computer Communications Systems*, pages 245–258, North-Holland, Amsterdam, 1984.

[19] R.M. Newman and J.L. Hullet. Distributed queueing: A fast and efficient packet access protocol for QPSX. In *Proc. 8th Intl. Conf. Computer Commun.*, pages 294–299, Munich, 1986.

[20] J.R. Pierce. Network for block switching of data. *Bell System Technical Journal*, 51(6):1133–1145, 1972.

[21] T. Raith. Performance analysis of multibus interconnection networks in distributed systems. In *Teletraffic Issues - Proc. ITC 11*, pages 662–668, North-Holland, Amsterdam, 1985.

[22] C.C. Reames and M.T. Liu. A loop network for simulataneous transmission of variable-length messages. In *Proceedings of 2nd Annual Symposium on Computer Architecture*, pages 7–12, January 1975.

[23] F.E. Ross. FDDI - A tutorial. *IEEE Communications Magazine*, 24(5):10–17, May 1986.

[24] H. Salwen. In praise of ring architecture for local area networks. In *Computer Design*, March 1983.

[25] I.M. Soi and K.K. Aggarwal. A review of computer communication network classification schemes. *IEEE Communication Magazine*, 19(3):24–32, March 1981.

[26] W.T. Strayer. A study of preemptable vs. non-preemptable token reservation access protocols. *ACM SIGCOMM Computer Communication Review*, pages 71–80, April 1991.

[27] S. Temple. The design of the Cambridge Fast Ring. In I.N. Dallas and E.B. Spratt, editors, *Ring Technology Local Area Networks*, pages 74–88, North-Holland, 1984.

[28] M. Xu and J.H. Herzog. Concurrent token ring protocol. In *1988 IEEE INFOCOM*, pages 145–154, 1988.